```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
import javax.swing.border.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class AxisDialog extends JDialog
{
  private JPanel panel1 = new JPanel();
  private BorderLayout borderLayout1 = new BorderLayout();

  double[][]  nomoRelative;
  String[]  nomoLabel = { "-1", "-.5", "-.2", "-.1", "-.05", "-.02", "-.01", "0",
".01", ".02", ".05", ".1", ".2", ".5", "1", "2", "5", "10", "20", "50",  "100",
"200", "500" };

  private JPanel jPanel1 = new JPanel();
  private JLabel jLabel1 = new JLabel();
  private JLabel jLabel2 = new JLabel();
  private GridLayout gridLayout1 = new GridLayout();
  private JPanel jPanel2 = new JPanel();
  private JComponent axisComponent = new JComponent ()
  {
    public void  paintComponent (
        Graphics  graphics )
    {
      Insets  inset = getInsets();
      int  w = getWidth() - inset.left - inset.right;
      int  h = getHeight() - inset.top - inset.bottom;
      int x0 = inset.left + w/3;
      int x1 = inset.left + 2*w/3;
      graphics.setColor( getBackground() );
      graphics.fillRect( inset.left, inset.top, w, h );
      graphics.setColor( getForeground() );
      graphics.drawLine( x0, inset.top, x0, inset.top + h - 1 );
      graphics.drawLine( x1, inset.top, x1, inset.top + h - 1 );
      if (nomoRelative == null)
        return;
      int  y0_clip = inset.top, y1_clip = inset.top;
      int  ascent = graphics.getFontMetrics().getAscent();
      int  em = graphics.getFontMetrics().getMaxAdvance();
      for (int  i = nomoRelative.length - 1;  i >= 0 ;  --i)
      {
        int  y0 = inset.top + h - (int) Math.round( nomoRelative[i][0] * h );
        int  y1 = inset.top + h - (int) Math.round( nomoRelative[i][1] * h );
        graphics.drawLine( x0, y0, x1, y1 );
        Rectangle2D  rect = graphics.getFontMetrics().getStringBounds( nomoLabel[i],
graphics );
        if (y0_clip < y0 - ascent)
        {
          graphics.drawString( nomoLabel[i], x0 - em/4 - (int) rect.getWidth(), y0
);
          graphics.drawLine( x0, y0, x0 - em/8, y0 );
```

```
        y0_clip = y0;
      }
      if (y1_clip < y1 - ascent)
      {
        graphics.drawString( nomoLabel[i], x1 + em/4, y1 );
        graphics.drawLine( x1, y1, x1 + em/8, y1 );
        y1_clip = y1;
      }
    }
  }
};
private BorderLayout borderLayout2 = new BorderLayout();
private JLabel jLabel3 = new JLabel();
private Border border1;
public AxisDialog(Frame frame, String title, boolean modal)
{
  super(frame, title, modal);
  try
  {
    jbInit();
    pack();
  }
  catch(Exception ex)
  {
    ex.printStackTrace();
  }
}

public AxisDialog()
{
  this(null, "", false);
}
private void jbInit() throws Exception
{
  border1 =
BorderFactory.createCompoundBorder(BorderFactory.createBevelBorder(BevelBorder.LOWER
ED,Color.white,Color.white,new Color(103, 101, 98),new Color(148, 145,
140)),BorderFactory.createEmptyBorder(6,6,6,6));
    panel1.setLayout(borderLayout1);
    jLabel1.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel1.setText("FlowJo");
    jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel2.setText("Logicle");
    jPanel1.setLayout(gridLayout1);
    axisComponent.setBackground(Color.white);
    axisComponent.setMinimumSize(new Dimension(60, 120));
    axisComponent.setPreferredSize(new Dimension(180, 360));
    jPanel2.setLayout(borderLayout2);
    jPanel1.setMinimumSize(new Dimension(60, 17));
    jPanel1.setPreferredSize(new Dimension(180, 17));
    gridLayout1.setColumns(3);
    panel1.setBorder(border1);
    jPanel2.setBackground(Color.white);
    getContentPane().add(panel1);
    panel1.add(jPanel1,  BorderLayout.NORTH);
    jPanel1.add(jLabel1, null);
    jPanel1.add(jLabel3, null);
    jPanel1.add(jLabel2, null);
    panel1.add(jPanel2,  BorderLayout.CENTER);
    jPanel2.add(axisComponent, BorderLayout.CENTER);
  }
}
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
import javax.swing.border.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class AxisDialog extends JDialog
{
  private JPanel panel1 = new JPanel();
  private BorderLayout borderLayout1 = new BorderLayout();

  double[][]  nomoRelative;
  String[]  nomoLabel = { "-1", "-.5", "-.2", "-.1", "-.05", "-.02", "-.01", "0",
".01", ".02", ".05", ".1", ".2", ".5", "1", "2", "5", "10", "20", "50",  "100",
"200", "500" };

   private JPanel jPanel1 = new JPanel();
   private JLabel jLabel1 = new JLabel();
   private JLabel jLabel2 = new JLabel();
   private GridLayout gridLayout1 = new GridLayout();
   private JPanel jPanel2 = new JPanel();
   private JComponent axisComponent = new JComponent ()
   {
     public void  paintComponent (
         Graphics  graphics )
     {
       Insets  inset = getInsets();
       int  w = getWidth() - inset.left - inset.right;
       int  h = getHeight() - inset.top - inset.bottom;
       int x0 = inset.left + w/3;
       int x1 = inset.left + 2*w/3;
       graphics.setColor( getBackground() );
       graphics.fillRect( inset.left, inset.top, w, h );
       graphics.setColor( getForeground() );
       graphics.drawLine( x0, inset.top, x0, inset.top + h - 1 );
       graphics.drawLine( x1, inset.top, x1, inset.top + h - 1 );
       if (nomoRelative == null)
         return;
       int  y0_clip = inset.top, y1_clip = inset.top;
       int  ascent = graphics.getFontMetrics().getAscent();
       int  em = graphics.getFontMetrics().getMaxAdvance();
       for (int  i = nomoRelative.length - 1;  i >= 0 ;  --i)
       {
         int  y0 = inset.top + h - (int) Math.round( nomoRelative[i][0] * h );
         int  y1 = inset.top + h - (int) Math.round( nomoRelative[i][1] * h );
         graphics.drawLine( x0, y0, x1, y1 );
         Rectangle2D  rect = graphics.getFontMetrics().getStringBounds( nomoLabel[i],
graphics );
         if (y0_clip < y0 - ascent)
         {
           graphics.drawString( nomoLabel[i], x0 - em/4 - (int) rect.getWidth(), y0
);
           graphics.drawLine( x0, y0, x0 - em/8, y0 );
```

```
        y0_clip = y0;
      }
      if (y1_clip < y1 - ascent)
      {
        graphics.drawString( nomoLabel[i], x1 + em/4, y1 );
        graphics.drawLine( x1, y1, x1 + em/8, y1 );
        y1_clip = y1;
      }
    }
  }
};
private BorderLayout borderLayout2 = new BorderLayout();
private JLabel jLabel3 = new JLabel();
private Border border1;
public AxisDialog(Frame frame, String title, boolean modal)
{
  super(frame, title, modal);
  try
  {
    jbInit();
    pack();
  }
  catch(Exception ex)
  {
    ex.printStackTrace();
  }
}

public AxisDialog()
{
  this(null, "", false);
}
private void jbInit() throws Exception
{
  border1 =
BorderFactory.createCompoundBorder(BorderFactory.createBevelBorder(BevelBorder.LOWER
ED,Color.white,Color.white,new Color(103, 101, 98),new Color(148, 145,
140)),BorderFactory.createEmptyBorder(6,6,6,6));
    panel1.setLayout(borderLayout1);
    jLabel1.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel1.setText("FlowJo");
    jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel2.setText("Logicle");
    jPanel1.setLayout(gridLayout1);
    axisComponent.setBackground(Color.white);
    axisComponent.setMinimumSize(new Dimension(60, 120));
    axisComponent.setPreferredSize(new Dimension(180, 360));
    jPanel2.setLayout(borderLayout2);
    jPanel1.setMinimumSize(new Dimension(60, 17));
    jPanel1.setPreferredSize(new Dimension(180, 17));
    gridLayout1.setColumns(3);
    panel1.setBorder(border1);
    jPanel2.setBackground(Color.white);
    getContentPane().add(panel1);
    panel1.add(jPanel1,  BorderLayout.NORTH);
    jPanel1.add(jLabel1, null);
    jPanel1.add(jLabel3, null);
    jPanel1.add(jLabel2, null);
    panel1.add(jPanel2,  BorderLayout.CENTER);
    jPanel2.add(axisComponent, BorderLayout.CENTER);
  }
}
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame extends JFrame {
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton jButton1 = new JButton();
  private JButton jButton2 = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private JLabel statusBar = new JLabel();
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrum_table = new JTable();
  private JMenuItem menuSpectrum = new JMenuItem();
  private JFileChooser  spectrumChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );
  private JMenuItem jMenuItem1 = new JMenuItem();
  private JMenuItem jMenuItem2 = new JMenuItem();

  //Construct the frame
  public ConverterFrame()
  {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
```

```
  }
  //Component initialization
  private void jbInit() throws Exception  {
    image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
    image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
    image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Log Like Data Converter");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
      }
    });
    jButton1.setIcon(image1);
    jButton1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    jButton1.setToolTipText("Open File");
    jButton2.setIcon(image2);
    jButton2.setToolTipText("Close File");
    jButton3.setIcon(image3);
    jButton3.setToolTipText("Help");
    menuSpectrum.setText("Spectrum");
    menuSpectrum.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        spectrum_action(e);
      }
    });
    jMenuItem1.setText("Data");
    jMenuItem1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    jMenuItem2.setText("Save");
    jToolBar.add(jButton1);
    jToolBar.add(jButton2);
```

```
  jToolBar.add(jButton3);
  jMenuFile.add(menuSpectrum);
  jMenuFile.add(jMenuItem1);
  jMenuFile.add(jMenuItem2);
  jMenuFile.add(jMenuFileExit);
  jMenuHelp.add(jMenuHelpAbout);
  jMenuBar1.add(jMenuFile);
  jMenuBar1.add(jMenuHelp);
  this.setJMenuBar(jMenuBar1);
  contentPane.add(jToolBar,  BorderLayout.NORTH);
  contentPane.add(statusBar, BorderLayout.SOUTH);
  contentPane.add(jScrollPane1, BorderLayout.CENTER);
  jScrollPane1.getViewport().add(spectrum_table, null);
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
}
//Help | About action performed
public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
}
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
  }
}

private static final String[]  stringArray = new String[0];
private FCSFile  fcs;
private String[]  sensor_names;
private int[][]  int_data;
private double[][]  raw_data;
private double[][]  compensated_data;
private double[][]  compensation_matrix;

void  compensate ()
throws IOException
{
  int  n = fcs.getTotal();
  int  m = sensor_names.length;

  raw_data = new double[ m ][ n ];
  compensated_data = new double[ m ][ n ];
  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensor_names[i] );
    int[]  input_data = int_data[p.getIndex()-1];
    double[]  output_data = raw_data[i];
    if (p.isLog())
    {
      double[]  antilog = new double[ p.getRange() ];
      antilog[0] = p.getMinimum();
      interpolate( antilog, 0, p.getRange(), p.getMaximum() );
      for (int  j = 0;  j < n;  ++i)
        output_data[j] = antilog[ input_data[j] ];
    }
    else
    {
      double  g = p.getGain();
      for (int  j = 0;  j < n;  ++i)
        output_data[j] = g * input_data[j];
```

```
    }
  }

  for (int  i = 0;  i < m;  ++i)
  {
    double[]  mult = compensation_matrix[i];
    double[]  output_data = compensated_data[i];
    for (int  j = 0;  j < n;  ++j)
    {
      double  x = 0;
      for (int  k = 0; k < m; ++k)
        x += raw_data[k][j] * mult[k];
      output_data[j] = x;
    }
  }

  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensor_names[i] );
    double[]  input_data = compensated_data[i];
    int[]  transformed_data = new int[n];
    if (p.isLog())
    {
      double[]  lookup = new double[ p.getRange() ];
      lookup[0] = p.getMinimum();
      interpolate( lookup, 0, p.getRange(), p.getMaximum() );
      for (int  j = 0;  j < n;  ++i)
      {
        int  k = Arrays.binarySearch( lookup, input_data[j] );
        if (k < 0)
          transformed_data[j] = -k;
        else
          transformed_data[j] = k;
      }
    }
    else
    {
      double  g = p.getGain();
      for (int  j = 0;  j < n;  ++i)
        transformed_data[j] = (int) Math.floor( input_data[j] / g );
    }
  }
}

void spectrum_action (
    ActionEvent e )
{
  int returnVal = spectrumChooser.showOpenDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      BufferedReader  br = new BufferedReader( new FileReader(
spectrumChooser.getSelectedFile() ) );
      br.readLine();
      br.readLine();
      StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
      ArrayList  al = new ArrayList();
      while (st.hasMoreTokens())
        al.add( st.nextToken() );
      String[]  columns = sensor_names = (String[]) al.toArray( stringArray );
      Double[][]  rows = new Double[sensor_names.length + 1][columns.length];
      for (int  r = 0;  r < sensor_names.length;  ++r)
```

```
        {
          st = new StringTokenizer( br.readLine(), "\t" );
          for (int  c = 0;  c < columns.length;  ++c)
          {
            rows[r][c] = new Double( Double.parseDouble( st.nextToken() ) );
          }
        }
/*
        for (int  c = 0;  c < columns.length;  ++c)
          rows[ sensor_names.length ][c] = new Double( 0 );
*/
        spectrum_table.setModel( new DefaultTableModel( rows, columns ) );
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }

    compensation_matrix = new double[ sensor_names.length ][ sensor_names.length ];
    for (int  i = 0;  i < sensor_names.length;  ++i)
      compensation_matrix[i][i] = 1.0;
  }

  private void  interpolate (
      double[]  func,
      int  i,
      int  n,
      double  x )
  {
    int  m = n / 2;
    double  y = Math.sqrt( func[i] * x );

    func[ i + m ] = y;

    if (m > 1 )
    {
      interpolate( func, i, m, y );
      interpolate( func, i + m, m, x );
    }

  }

  void data_action(ActionEvent e)
  {
    int returnVal = spectrumChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        FCSFile  fcs = new FCSFile( spectrumChooser.getSelectedFile() );
        int  n = fcs.getTotal();
        int  m = fcs.getParameters();
        int[][]  int_value = new int[m][n];
        FCSHandler  lmi = fcs.getInputIterator();
        for (int  i = 0;  lmi.hasMoreEvents();  ++i)
          for (int  j = 0;  lmi.hasMoreValues();  ++j)
            int_value[j][i] = lmi.readValue();
        this.fcs = fcs;
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
```

```
        }
      }
    }
  }
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame extends JFrame {
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton jButton1 = new JButton();
  private JButton jButton2 = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private JLabel statusBar = new JLabel();
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrum_table = new JTable();
  private JMenuItem menuSpectrum = new JMenuItem();
  private JFileChooser  spectrumChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );
  private JMenuItem jMenuItem1 = new JMenuItem();
  private JMenuItem jMenuItem2 = new JMenuItem();

  //Construct the frame
  public ConverterFrame()
  {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
```

```
  }
  //Component initialization
  private void jbInit() throws Exception  {
    image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
    image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
    image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Log Like Data Converter");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
      }
    });
    jButton1.setIcon(image1);
    jButton1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    jButton1.setToolTipText("Open File");
    jButton2.setIcon(image2);
    jButton2.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        save_action(e);
      }
    });
    jButton2.setToolTipText("Close File");
    jButton3.setIcon(image3);
    jButton3.setToolTipText("Help");
    menuSpectrum.setText("Spectrum");
    menuSpectrum.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        spectrum_action(e);
      }
    });
    jMenuItem1.setText("Data");
    jMenuItem1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
```

```
    {
      data_action(e);
    }
  });
  jMenuItem2.setText("Save");
  jMenuItem2.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      save_action(e);
    }
  });
  jToolBar.add(jButton1);
  jToolBar.add(jButton2);
  jToolBar.add(jButton3);
  jMenuFile.add(menuSpectrum);
  jMenuFile.add(jMenuItem1);
  jMenuFile.add(jMenuItem2);
  jMenuFile.add(jMenuFileExit);
  jMenuHelp.add(jMenuHelpAbout);
  jMenuBar1.add(jMenuFile);
  jMenuBar1.add(jMenuHelp);
  this.setJMenuBar(jMenuBar1);
  contentPane.add(jToolBar,  BorderLayout.NORTH);
  contentPane.add(statusBar, BorderLayout.SOUTH);
  contentPane.add(jScrollPane1, BorderLayout.CENTER);
  jScrollPane1.getViewport().add(spectrum_table, null);
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
}
//Help | About action performed
public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
}
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
  }
}

private static final String[]  stringArray = new String[0];
private FCSFile  fcs;
private String[]  sensor_names;
private int[][]  int_data;
private int[][]  new_data;
private double[][]  raw_data;
private double[][]  compensated_data;
private double[][]  compensation_matrix;

void  compensate ()
throws IOException
{
  int  n = fcs.getTotal();
  int  m = sensor_names.length;

  raw_data = new double[ m ][ n ];
  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensor_names[i] );
    int[]  input_data = int_data[p.getIndex()-1];
```

```
    double[]  output_data = raw_data[i];
    if (p.isLog())
    {
      double[]  antilog = new double[ p.getRange() ];
      antilog[0] = p.getMinimum();
      interpolate( antilog, 0, p.getRange(), p.getMaximum() );
      for (int  j = 0;  j < n;  ++j)
        output_data[j] = antilog[ input_data[j] ];
    }
    else
    {
      double  g = p.getGain();
      for (int  j = 0;  j < n;  ++i)
        output_data[j] = g * input_data[j];
    }
  }

  compensated_data = new double[ m ][ n ];
  for (int  i = 0;  i < m;  ++i)
  {
    double[]  mult = compensation_matrix[i];
    double[]  output_data = compensated_data[i];
    for (int  j = 0;  j < n;  ++j)
    {
      double  x = 0;
      for (int  k = 0; k < m; ++k)
        x += raw_data[k][j] * mult[k];
      output_data[j] = x;
    }
  }

  new_data = new int[ m ][ n ];
  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensor_names[i] );
    double[]  input_data = compensated_data[i];
    int[]  transformed_data = new_data[i];
    if (p.isLog())
    {
      double[]  lookup = new double[ p.getRange() ];
      lookup[0] = p.getMinimum();
      interpolate( lookup, 0, p.getRange(), p.getMaximum() );
      for (int  j = 0;  j < n;  ++j)
      {
        int  k = Arrays.binarySearch( lookup, input_data[j] );
        if (k < 0)
          transformed_data[j] = -k;
        else
          transformed_data[j] = k;
      }
    }
    else
    {
      double  g = p.getGain();
      for (int  j = 0;  j < n;  ++i)
        transformed_data[j] = (int) Math.floor( input_data[j] / g );
    }
  }
}

void spectrum_action (
    ActionEvent e )
{
```

```
    int returnVal = spectrumChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        BufferedReader  br = new BufferedReader( new FileReader(
spectrumChooser.getSelectedFile() ) );
        br.readLine();
        br.readLine();
        StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
        ArrayList  al = new ArrayList();
        while (st.hasMoreTokens())
          al.add( st.nextToken() );
        String[]  columns = sensor_names = (String[]) al.toArray( stringArray );
        Double[][]  rows = new Double[sensor_names.length + 1][columns.length];
        for (int  r = 0;  r < sensor_names.length;  ++r)
        {
          st = new StringTokenizer( br.readLine(), "\t" );
          for (int  c = 0;  c < columns.length;  ++c)
          {
            rows[r][c] = new Double( Double.parseDouble( st.nextToken() ) );
          }
        }
        spectrum_table.setModel( new DefaultTableModel( rows, columns ) );
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }

    compensation_matrix = new double[ sensor_names.length ][ sensor_names.length ];
    for (int  i = 0;  i < sensor_names.length;  ++i)
      compensation_matrix[i][i] = 1.0;
  }

  private void  interpolate (
      double[]  func,
      int  i,
      int  n,
      double  x )
  {
    int  m = n / 2;
    double  y = Math.sqrt( func[i] * x );

    func[ i + m ] = y;

    if (m > 1 )
    {
      interpolate( func, i, m, y );
      interpolate( func, i + m, m, x );
    }

  }

  void data_action(ActionEvent e)
  {
    int returnVal = spectrumChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        FCSFile  fcs = new FCSFile( spectrumChooser.getSelectedFile() );
```

```
      int  n = fcs.getTotal();
      int  m = fcs.getParameters();
      int[][]  int_value = new int[m][n];
      FCSHandler  lmi = fcs.getInputIterator();
      for (int  i = 0;  lmi.hasMoreEvents();  ++i)
        for (int  j = 0;  lmi.hasMoreValues();  ++j)
          int_value[j][i] = lmi.readValue();
      this.int_data = int_value;
      this.fcs = fcs;
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
}

void save_action(ActionEvent e)
{
  int returnVal = spectrumChooser.showSaveDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      compensate();
      fcs.setFile( spectrumChooser.getSelectedFile() );
      for (int  i = 0;  i < sensor_names.length;  ++i)
      {
        FCSParameter  oldp = fcs.getParameter( sensor_names[i] );
        FCSParameter  newp = fcs.addParameter();
        newp.setBits( oldp.getBits() );
        newp.setRange( oldp.getRange() );
        if (oldp.isLog())
        {
          newp.setDecades( oldp.getDecades() );
          newp.setMinimum( oldp.getMinimum() );
        }
        else
        {
          newp.setDecades( Double.NaN );
          newp.setGain( oldp.getGain() );
        }

      }
      int  n = fcs.getTotal();
      int  m = fcs.getParameters();
      FCSHandler  lmi = fcs.getOutputIterator();
      for (int  i = 0;  lmi.hasMoreEvents();  ++i)
      {
        for (int  j = 0;  j < int_data.length;  ++j)
          lmi.writeValue( int_data[j][i] );
      }
      lmi.close();
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
 }
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame extends JFrame {
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton jButton1 = new JButton();
  private JButton jButton2 = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private JLabel statusBar = new JLabel();
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrum_table = new JTable();
  private JMenuItem menuSpectrum = new JMenuItem();
  private JFileChooser  fileChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );
  private JMenuItem jMenuItem1 = new JMenuItem();
  private JMenuItem jMenuItem2 = new JMenuItem();

  //Construct the frame
  public ConverterFrame()
  {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
```

```
  }
  //Component initialization
  private void jbInit() throws Exception  {
    image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
    image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
    image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Log Like Data Converter");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
      }
    });
    jButton1.setIcon(image1);
    jButton1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    jButton1.setToolTipText("Open File");
    jButton2.setIcon(image2);
    jButton2.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        save_action(e);
      }
    });
    jButton2.setToolTipText("Close File");
    jButton3.setIcon(image3);
    jButton3.setToolTipText("Help");
    menuSpectrum.setText("Spectrum");
    menuSpectrum.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        spectrum_action(e);
      }
    });
    jMenuItem1.setText("Data");
    jMenuItem1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
```

```
    {
      data_action(e);
    }
  });
  jMenuItem2.setText("Save");
  jMenuItem2.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      save_action(e);
    }
  });
  functionChoice.setModel(functionModel);
  jToolBar.add(jButton1);
  jToolBar.add(jButton2);
  jToolBar.add(jButton3);
  jToolBar.add(functionChoice, null);
  jMenuFile.add(menuSpectrum);
  jMenuFile.add(jMenuItem1);
  jMenuFile.add(jMenuItem2);
  jMenuFile.add(jMenuFileExit);
  jMenuHelp.add(jMenuHelpAbout);
  jMenuBar1.add(jMenuFile);
  jMenuBar1.add(jMenuHelp);
  this.setJMenuBar(jMenuBar1);
  contentPane.add(jToolBar,  BorderLayout.NORTH);
  contentPane.add(statusBar, BorderLayout.SOUTH);
  contentPane.add(jScrollPane1, BorderLayout.CENTER);
  jScrollPane1.getViewport().add(spectrum_table, null);
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
}
//Help | About action performed
public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
}
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
  }
}

private static final String[]  stringArray = new String[0];
private FCSFile  fcs;
private String[]  sensor_names;
private int[][]  int_data;
private int[][]  new_data;
private double[][]  raw_data;
private double[][]  compensated_data;
private double[][]  compensation_matrix;
private TableModel  model_parameters;
private JComboBox functionChoice = new JComboBox();
private String[]  functions = { "log to linear splice", "x + log x" };
private DefaultComboBoxModel functionModel = new DefaultComboBoxModel( functions
);

  private double  getModelParameter (
      int  row,
      int  col )
  {
```

```
Object  obj = model_parameters.getValueAt( row, col );
if (obj == null)
  return  Double.NaN;
if (obj instanceof Number)
 return ((Number) obj).doubleValue();
if (obj instanceof String)
{
  try
  {
    return  Double.parseDouble( (String) obj );
  }
  catch (NumberFormatException ex)
  {
    return  Double.NaN;
  }
 }
 else
 {
   model_parameters.setValueAt( null, row, col );
   return  Double.NaN;
 }
}

private void  invert (
    double[][]  matrix )
{
  int  row = 0, col = 0, n = matrix.length;
  int  pivot[] = new int[n];
  int  row_index[] = new int[n];
  int  col_index[] = new int[n];

  for (int  i = 0;  i < n;  ++i)
  {
    double  big = 0;
    for (int  j = 0;  j < n;  ++j)
    {
      if (pivot[j] != 1)
        for (int  k = 0;  k < n;  ++k)
        {
          if (pivot[k] == 0)
          {
            double  abs = Math.abs(matrix[j][k]);
            if (abs >= big)
            {
              big = abs;
              row = j;
              col = k;
            }
          }
          else if (pivot[k] > 1)
            throw  new IllegalArgumentException( "matrix is singular" );
        }
      }
      ++pivot[col];
      row_index[i] = row;
      col_index[i] = col;

      if (row != col)
      {
        System.out.println( "pivoting!" );
        for (int  k = 0;  k < n;  ++k)
        {
          double  t = matrix[row][k];
```

```
        matrix[row][k] = matrix[col][k];
        matrix[col][k] = t;
      }
    }

    if (matrix[col][col] == 0)
      throw  new IllegalArgumentException( "matrix is singular" );
    double inverse = 1/matrix[col][col];
    matrix[col][col] = 1;
    for (int  j = 0;  j < n;  ++j)
      matrix[col][j] *= inverse;
    for (int  j = 0;  j < n;  ++j)
      if (j != col)
      {
        double  t = matrix[j][col];
        matrix[j][col] = 0;
        for (int  k = 0;  k < n;  ++k)
          matrix[j][k] -= matrix[col][k] * t;
      }
  }

  for (int  i = n-1;  i >= 0;  --i)
    if (row_index[i] != col_index[i])
      for (int  j = 0;  j < n;  ++j)
      {
        double  t = matrix[j][row_index[i]];
        matrix[j][row_index[i]] = matrix[j][col_index[i]];
        matrix[j][col_index[i]] = t;
      }
}

void  compensate ()
throws IOException
{
  int  n = fcs.getTotal();
  int  m = sensor_names.length;

  double[][]  compensation_matrix = new double[m][m];
  for (int  i = 0;  i < m;  ++i)
    compensation_matrix[i][i] = 1.0;
  for (int  i = 0;  i < m; ++i)
    for (int  j = 0;  j < m;  ++j)
    {
      double  coefficient = getModelParameter( i, j );
      if (Double.isNaN( coefficient ))
        coefficient = 0;
      compensation_matrix[j][i] = coefficient;
    }
  invert( compensation_matrix );

  raw_data = new double[ m ][ n ];
  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensor_names[i] );
    int[]  input_data = int_data[p.getIndex()-1];
    double[]  output_data = raw_data[i];
    if (p.isLog())
    {
      double[]  antilog = new double[ p.getRange() ];
      antilog[0] = p.getMinimum();
      interpolate( antilog, 0, p.getRange(), p.getMaximum() );
      for (int  j = 0;  j < n;  ++j)
        output_data[j] = antilog[ input_data[j] ];
```

```
    }
    else
    {
      double  g = p.getGain();
      for (int  j = 0;  j < n;  ++i)
        output_data[j] = input_data[j] / g;
    }
  }

  compensated_data = new double[ m ][ n ];
  for (int  i = 0;  i < m;  ++i)
  {
    double[]  output_data = compensated_data[i];
    for (int  j = 0;  j < m;  ++j)
    {
      double[]  input_data = raw_data[j];
      double  mult = compensation_matrix[i][j];
      for (int  k = 0;  k < n;  ++k)
        output_data[k] += input_data[k] * mult;
    }
  }

  new_data = new int[ m ][ n ];
  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensor_names[i] );
    double[]  input_data = compensated_data[i];
    int[]  transformed_data = new_data[i];
    if (p.isLog())
    {
      double[]  lookup = new double[ p.getRange() ];
      lookup[0] = p.getMinimum();
      interpolate( lookup, 0, p.getRange(), p.getMaximum() );
      double  parameter = getModelParameter( m, i );
      if (!Double.isNaN( parameter ))
      {
        switch (functionChoice.getSelectedIndex())
        {
          case 0:
            int  splice = (int) Math.round(
              (Math.log( parameter ) - Math.log( p.getMinimum() )) * (double)
p.getRange() / p.getDecades() / Math.log( 10 ) );
            if (splice > 0 && splice < p.getRange() - 1)
            {
              double  delta = (lookup[splice+1] - lookup[splice-1])/2;
              double  base = lookup[splice];
              for (int  j = 0;  j < splice;  ++j)
                lookup[j] = base - (splice - j) * delta;
            }
            break;

          case 1:
            parameter *= p.getDecades() / p.getRange();
            for (int  j = 0;  j < lookup.length;  ++j)
              lookup[j] += parameter * j;
            break;
        }
      }
      for (int  j = 0;  j < n;  ++j)
      {
        int  k = Arrays.binarySearch( lookup, input_data[j] );
        if (k < 0)
          transformed_data[j] = -k;
```

```
        else
          transformed_data[j] = k;
      }
    }
    else
    {
      double  g = p.getGain();
      for (int  j = 0;  j < n;  ++i)
        transformed_data[j] = (int) Math.round( input_data[j] * g );
    }
  }
}

void spectrum_action (
    ActionEvent e )
{
  fileChooser.setDialogTitle( "Load Spectrum File" );
  int returnVal = fileChooser.showOpenDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      BufferedReader  br = new BufferedReader( new FileReader(
fileChooser.getSelectedFile() ) );
      br.readLine();
      br.readLine();
      StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
      ArrayList  al = new ArrayList();
      while (st.hasMoreTokens())
        al.add( st.nextToken() );
      String[]  columns = sensor_names = (String[]) al.toArray( stringArray );
      Double[][]  rows = new Double[sensor_names.length + 1][columns.length];
      for (int  r = 0;  r < sensor_names.length;  ++r)
      {
        st = new StringTokenizer( br.readLine(), "\t" );
        for (int  c = 0;  c < columns.length;  ++c)
        {
          rows[r][c] = new Double( Double.parseDouble( st.nextToken() ) );
        }
      }
      model_parameters = new DefaultTableModel( rows, columns );
      spectrum_table.setModel( model_parameters );
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
}

private void  interpolate (
    double[]  func,
    int  i,
    int  n,
    double  x )
{
  int  m = n / 2;
  double  y = Math.sqrt( func[i] * x );

  func[ i + m ] = y;

  if (m > 1 )
  {
```

```
    interpolate( func, i, m, y );
    interpolate( func, i + m, m, x );
  }

}

void data_action(ActionEvent e)
{
  fileChooser.setDialogTitle( "Load FCS Data File" );
  int returnVal = fileChooser.showOpenDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      File  f = fileChooser.getSelectedFile();
      FCSFile  fcs = new FCSFile( f );
      int  n = fcs.getTotal();
      int  m = fcs.getParameters();
      int[][]  int_value = new int[m][n];
      FCSHandler  lmi = fcs.getInputIterator();
      for (int  i = 0;  lmi.hasMoreEvents();  ++i)
        for (int  j = 0;  lmi.hasMoreValues();  ++j)
          int_value[j][i] = lmi.readValue();
      this.int_data = int_value;
      this.fcs = fcs;
      statusBar.setText( f.getName() );
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
}

void save_action(ActionEvent e)
{
  fileChooser.setDialogTitle( "Save New FCS Data File" );
  int returnVal = fileChooser.showSaveDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      compensate();
      for (int  i = 0;  i < sensor_names.length;  ++i)
      {
        FCSParameter  oldp = fcs.getParameter( sensor_names[i] );
        FCSParameter  newp = fcs.addParameter();
        newp.setAttribute( "$P", "N", "["+sensor_names[i]+"]" );
        newp.setAttribute( "$P", "S", oldp.getAttribute( "$P", "S" ) );
        newp.setBits( oldp.getBits() );
        newp.setRange( oldp.getRange() );
        newp.setLog( oldp.isLog() );
        newp.setDecades( oldp.getDecades() );
        newp.setMinimum( oldp.getMinimum() );
        newp.setGain( oldp.getGain() );
      }

      fcs.setFile( fileChooser.getSelectedFile() );
      FCSHandler  lmi = fcs.getOutputIterator();
      for (int  i = 0;  lmi.hasMoreEvents();  ++i)
      {
        for (int  j = 0;  j < int_data.length;  ++j)
          lmi.writeValue( int_data[j][i] );
```

```
            for (int  j = 0;  j < new_data.length;  ++j)
              lmi.writeValue( new_data[j][i] );
          }
          lmi.close();
        }
        catch (Exception ex)
        {
          ex.printStackTrace();
        }
      }
    }
  }
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;
import javax.swing.event.TableModelListener;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame
extends JFrame
{
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton jButton1 = new JButton();
  private JButton jButton2 = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private JLabel statusBar = new JLabel();
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrumTable = new JTable();
  private JMenuItem menuSpectrum = new JMenuItem();
  private JFileChooser  fileChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );
  private JMenuItem jMenuItem1 = new JMenuItem();
  private JMenuItem jMenuItem2 = new JMenuItem();

  //Construct the frame
  public ConverterFrame()
  {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
      spectrumTable.setModel( modelParameters );
    }
```

```
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }
  //Component initialization
  private void jbInit() throws Exception  {
    image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
    image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
    image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Logicle Data Converter");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
      }
    });
    jButton1.setIcon(image1);
    jButton1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    jButton1.setToolTipText("Open File");
    jButton2.setIcon(image2);
    jButton2.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        save_action(e);
      }
    });
    jButton2.setToolTipText("Close File");
    jButton3.setIcon(image3);
    jButton3.setToolTipText("Help");
    menuSpectrum.setIcon(new
 ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
    menuSpectrum.setText("Spectrum");
    menuSpectrum.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        spectrum_action(e);
```

```
    }
  });
  jMenuItem1.setIcon(image1);
  jMenuItem1.setText("Data");
  jMenuItem1.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      data_action(e);
    }
  });
  jMenuItem2.setIcon(image2);
  jMenuItem2.setText("Save");
  jMenuItem2.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      save_action(e);
    }
  });
  functionChoice.setModel(functionModel);
  jButton4.setMaximumSize(new Dimension(51, 27));
  jButton4.setMinimumSize(new Dimension(51, 27));
  jButton4.setIcon(new
ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
  jButton4.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      spectrum_action(e);
    }
  });
  jToolBar.add(jButton4, null);
  jToolBar.add(jButton1);
  jToolBar.add(jButton2);
  jToolBar.add(jButton3);
  jToolBar.add(functionChoice, null);
  jMenuFile.add(menuSpectrum);
  jMenuFile.add(jMenuItem1);
  jMenuFile.add(jMenuItem2);
  jMenuFile.add(jMenuFileExit);
  jMenuHelp.add(jMenuHelpAbout);
  jMenuBar1.add(jMenuFile);
  jMenuBar1.add(jMenuHelp);
  this.setJMenuBar(jMenuBar1);
  contentPane.add(jToolBar,  BorderLayout.NORTH);
  contentPane.add(statusBar, BorderLayout.SOUTH);
  contentPane.add(jScrollPane1, BorderLayout.CENTER);
  jScrollPane1.getViewport().add(spectrumTable, null);
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
}
//Help | About action performed
public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
}
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
  }
```

```
}

private static final String[]  stringArray = new String[0];
private FCSFile  fcs;
private String[]  sensorNames = new String[0];
private int[][]  originalData;
private int[][]  new_data;
private double[][]  raw_data;
private double[][]  compensated_data;
private double[][]  compensation_matrix;

private class  LogicleFunction
{
  public void  transform (
      int  column,
     FCSParameter  parameter )
  {
    if (parameter.isLog())
    {
      double[]  lookup = new double[ parameter.getRange() ];
      lookup[0] = parameter.getMinimum();
      interpolate( lookup, 0, parameter.getRange(), parameter.getMaximum() );
    }
  }
}

private  LogicleFunction  logLinearSplice = new LogicleFunction ()
{

  public void  transform (
      int  column,
     FCSParameter  parameter )
  {
    if (Double.isNaN(modelParameters.getParameter( sensorNames.length, column ))
|| !parameter.isLog())
    {
      super.transform( column, parameter );
      return;
    }

    double[]  lookup = new double[ parameter.getRange() ];
    lookup[0] = parameter.getMinimum();
    interpolate( lookup, 0, parameter.getRange(), parameter.getMaximum() );
    double  parameter1 = modelParameters.getParameter( sensorNames.length, 0 /* i
*/ );
    if (Double.isNaN( parameter1 ))
      ;// break;
    int  splice = (int) Math.round(
        (Math.log( parameter1 ) - Math.log( parameter.getMinimum() )) * (double)
parameter.getRange() / parameter.getDecades() / ln_10 );
    if (splice > 0 && splice < parameter.getRange() - 1)
    {
      double  delta = (lookup[splice+1] - lookup[splice-1])/2;
      double  base = lookup[splice];
      for (int  j = 0;  j < splice;  ++j)
        lookup[j] = base - (splice - j) * delta;
    }
  }
};

private class  ModelParameters
extends AbstractTableModel
{
```

```
private double[][]  parameters = new double[0][];

public int getRowCount()
{
  return  parameters.length;
}

public int getColumnCount()
{
  return  sensorNames.length;
}

public String getColumnName (
    int columnIndex )
{
  return  sensorNames[ columnIndex ];
}

public Class getColumnClass (
    int columnIndex )
{
  return  super.getColumnClass( columnIndex );
}

public boolean isCellEditable (
    int rowIndex,
    int columnIndex )
{
  return  true;
}

public Object getValueAt (
    int rowIndex,
    int columnIndex )
{
  double  value = parameters[ rowIndex ][ columnIndex ];
  if (Double.isNaN( value ))
    return  null;
  else
    return  String.valueOf( value );
}

public void setValueAt (
    Object aValue,
    int rowIndex,
    int columnIndex)
{
  double  dvalue;
  if (aValue instanceof Number)
    dvalue = ((Number) aValue).doubleValue();
  else if (aValue instanceof String)
  {
    try
    {
      dvalue = Double.valueOf( (String) aValue ).doubleValue();
    }
    catch (NumberFormatException ex)
    {
      dvalue = Double.NaN;
    }
  }
  else
    dvalue = Double.NaN;
```

```
      if (Double.isNaN( dvalue) && rowIndex < sensorNames.length)
        if ( rowIndex == columnIndex)
          dvalue = 1;
        else
          dvalue = 0;

      parameters[ rowIndex ][ columnIndex ] = dvalue;
      super.setValueAt( aValue, rowIndex, rowIndex );
    }

    public void  setParameters (
        double[][]  parameters )
    {
      this.parameters = parameters;
      this.fireTableStructureChanged();
    }

    public double  getParameter (
        int rowIndex,
        int columnIndex )
    {
      return  parameters[rowIndex][columnIndex];
    }
  };
  private ModelParameters  modelParameters = new ModelParameters();
  private JComboBox functionChoice = new JComboBox();
  private String[]  functions = { "log linear splice", "x + a log x + b",
"hyperbolic" };
  private DefaultComboBoxModel functionModel = new DefaultComboBoxModel( functions
);
  /*
  private double  getModelParameter (
      int  row,
      int  col )
  {
    Object  obj = modelParameters.getValueAt( row, col );
    if (obj == null)
      return  Double.NaN;
    if (obj instanceof Number)
     return ((Number) obj).doubleValue();
    if (obj instanceof String)
    {
      try
      {
        return  Double.parseDouble( (String) obj );
      }
      catch (NumberFormatException ex)
      {
        return  Double.NaN;
      }
    }
    else
    {
      modelParameters.setValueAt( null, row, col );
      return  Double.NaN;
    }
  }
 */
  private void  invert (
      double[][]  matrix )
  {
    int  row = 0, col = 0, n = matrix.length;
    int  pivot[] = new int[n];
```

```
int  row_index[] = new int[n];
int  col_index[] = new int[n];

for (int  i = 0;  i < n;  ++i)
{
  double  big = 0;
  for (int  j = 0;  j < n;  ++j)
  {
    if (pivot[j] != 1)
      for (int  k = 0;  k < n;  ++k)
      {
        if (pivot[k] == 0)
        {
          double  abs = Math.abs(matrix[j][k]);
          if (abs >= big)
          {
            big = abs;
            row = j;
            col = k;
          }
        }
        else if (pivot[k] > 1)
          throw  new IllegalArgumentException( "matrix is singular" );
      }
  }
  ++pivot[col];
  row_index[i] = row;
  col_index[i] = col;

  if (row != col)
  {
    System.out.println( "pivoting!" );
    for (int  k = 0;  k < n;  ++k)
    {
      double  t = matrix[row][k];
      matrix[row][k] = matrix[col][k];
      matrix[col][k] = t;
    }
  }

  if (matrix[col][col] == 0)
    throw  new IllegalArgumentException( "matrix is singular" );
  double inverse = 1/matrix[col][col];
  matrix[col][col] = 1;
  for (int  j = 0;  j < n;  ++j)
    matrix[col][j] *= inverse;
  for (int  j = 0;  j < n;  ++j)
    if (j != col)
    {
      double  t = matrix[j][col];
      matrix[j][col] = 0;
      for (int  k = 0;  k < n;  ++k)
        matrix[j][k] -= matrix[col][k] * t;
    }
}

for (int  i = n-1;  i >= 0;  --i)
  if (row_index[i] != col_index[i])
    for (int  j = 0;  j < n;  ++j)
    {
      double  t = matrix[j][row_index[i]];
      matrix[j][row_index[i]] = matrix[j][col_index[i]];
      matrix[j][col_index[i]] = t;
```

```
      }
}

private static final double  ln_10 = Math.log( 10 );
private JButton jButton4 = new JButton();

void  compensate ()
throws IOException
{
  int  n = fcs.getTotal();
  int  m = sensorNames.length;

  raw_data = new double[m][n];
  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensorNames[i] );
    int[]  input_data = originalData[p.getIndex()-1];
    double[]  output_data = raw_data[i];
    if (p.isLog())
    {
      double[]  antilog = new double[ p.getRange() ];
      antilog[0] = p.getMinimum();
      interpolate( antilog, 0, p.getRange(), p.getMaximum() );
      for (int  j = 0;  j < n;  ++j)
        output_data[j] = antilog[ input_data[j] ];
    }
    else
    {
      double  g = p.getGain();
      for (int  j = 0;  j < n;  ++i)
        output_data[j] = input_data[j] / g;
    }
  }

  double[][]  compensation_matrix = new double[m][m];
  for (int  i = 0;  i < m; ++i)
    for (int  j = 0;  j < m;  ++j)
      compensation_matrix[j][i] = modelParameters.getParameter( i, j );
  invert( compensation_matrix );

  compensated_data = new double[m][n];
  for (int  i = 0;  i < m;  ++i)
  {
    double[]  output_data = compensated_data[i];
    for (int  j = 0;  j < m;  ++j)
    {
      double[]  input_data = raw_data[j];
      double  mult = compensation_matrix[i][j];
      for (int  k = 0;  k < n;  ++k)
        output_data[k] += input_data[k] * mult;
    }
  }

  new_data = new int[m][n];
  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensorNames[i] );
    double[]  input_data = compensated_data[i];
    int[]  transformed_data = new_data[i];
    if (p.isLog())
    {
      double[]  lookup = new double[ p.getRange() ];
      lookup[0] = p.getMinimum();
```

```
        interpolate( lookup, 0, p.getRange(), p.getMaximum() );
        switch (functionChoice.getSelectedIndex())
        {
          case 0:
            lookup[0] = p.getMinimum();
            interpolate( lookup, 0, p.getRange(), p.getMaximum() );
            double  parameter1 = modelParameters.getParameter( m, i );
            if (Double.isNaN( parameter1 ))
              break;
            int  splice = (int) Math.round(
                (Math.log( parameter1 ) - Math.log( p.getMinimum() )) * (double)
p.getRange() / p.getDecades() / ln_10 );
            if (splice > 0 && splice < p.getRange() - 1)
            {
              double  delta = (lookup[splice+1] - lookup[splice-1])/2;
              double  base = lookup[splice];
              for (int  j = 0;  j < splice;  ++j)
                lookup[j] = base - (splice - j) * delta;
            }
            break;

          case 1:
            lookup[0] = p.getMinimum();
            interpolate( lookup, 0, p.getRange(), p.getMaximum() );
            parameter1 = modelParameters.getParameter( m, i );
            if (Double.isNaN( parameter1 ))
              break;
            double  parameter2 = modelParameters.getParameter( m + 1, i );
            if (Double.isNaN( parameter2 ))
              parameter2 = 0;
            parameter2 += parameter1 * Math.log( p.getMinimum() ) / ln_10;
            parameter1 *= p.getDecades() / p.getRange();
            for (int  j = 0;  j < lookup.length;  ++j)
              lookup[j] += parameter1 * j + parameter2;
            break;

          case 2:
            lookup[0] = 1;
            interpolate( lookup, 0, p.getRange(), Math.exp( p.getDecades() * ln_10 )
);
            parameter1 = modelParameters.getParameter( m, i );
            if (Double.isNaN( parameter1 ))
              parameter1 = 0;
            double  x = p.getMinimum();
            for (int  j = 0;  j < lookup.length;  ++j)
              lookup[j] = x * (lookup[j] - 1/lookup[j]) - parameter1;
            break;
        }
        for (int  j = 0;  j < n;  ++j)
        {
          int  k = Arrays.binarySearch( lookup, input_data[j] );
          if (k < 0)
            transformed_data[j] = -k;
          else
            transformed_data[j] = k;
        }
      }
      else
      {
        double  g = p.getGain();
        for (int  j = 0;  j < n;  ++i)
          transformed_data[j] = (int) Math.round( input_data[j] * g );
      }
```

```
    }
  }

  void spectrum_action (
      ActionEvent e )
  {
    fileChooser.setDialogTitle( "Load Spectrum File" );
    int returnVal = fileChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        BufferedReader  br = new BufferedReader( new FileReader(
fileChooser.getSelectedFile() ) );
        br.readLine();
        br.readLine();
        StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
        ArrayList  al = new ArrayList();
        while (st.hasMoreTokens())
          al.add( st.nextToken() );
        sensorNames = (String[]) al.toArray( stringArray );
        double[][]  parameters = new double[sensorNames.length +
2][sensorNames.length];
        int  r;
        for (r = 0;  r < sensorNames.length;  ++r)
        {
          st = new StringTokenizer( br.readLine(), "\t" );
          for (int  c = 0;  c < sensorNames.length;  ++c)
            parameters[r][c] =  Double.parseDouble( st.nextToken() );
        }
        for (;  r < parameters.length;  ++r)
          for (int  c = 0;  c < sensorNames.length;  ++c)
            parameters[r][c] =  Double.NaN;
        modelParameters.setParameters( parameters );

      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }

  private void  interpolate (
      double[]  func,
      int  i,
      int  n,
      double  x )
  {
    int  m = n / 2;
    double  y = Math.sqrt( func[i] * x );

    func[ i + m ] = y;

    if (m > 1 )
    {
      interpolate( func, i, m, y );
      interpolate( func, i + m, m, x );
    }

  }

  void data_action(ActionEvent e)
```

```
{
  fileChooser.setDialogTitle( "Load FCS Data File" );
  int returnVal = fileChooser.showOpenDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      File  f = fileChooser.getSelectedFile();
      FCSFile  fcs = new FCSFile( f );
      int  n = fcs.getTotal();
      int  m = fcs.getParameters();
      int[][]  int_value = new int[m][n];
      FCSHandler  lmi = fcs.getInputIterator();
      for (int  i = 0;  lmi.hasMoreEvents();  ++i)
        for (int  j = 0;  lmi.hasMoreValues();  ++j)
          int_value[j][i] = lmi.readValue();
      this.originalData = int_value;
      this.fcs = fcs;
      statusBar.setText( f.getName() );
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
}

void save_action(ActionEvent e)
{
  fileChooser.setDialogTitle( "Save New FCS Data File" );
  int returnVal = fileChooser.showSaveDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      compensate();

      FCSFile  nfcs = new FCSFile( fileChooser.getSelectedFile() );
      nfcs.getTextSegment().readFrom( fcs.getTextSegment() );
      for (int  i = 0;  i < sensorNames.length;  ++i)
      {
        FCSParameter  oldp = fcs.getParameter( sensorNames[i] );
        FCSParameter  newp = nfcs.addParameter();
        newp.setAttribute( "$P", "N", "["+sensorNames[i]+"]" );
        newp.setAttribute( "$P", "S", oldp.getAttribute( "$P", "S" ) );
        newp.setBits( oldp.getBits() );
        newp.setRange( oldp.getRange() );
        newp.setLog( oldp.isLog() );
        newp.setDecades( oldp.getDecades() );
        newp.setMinimum( oldp.getMinimum() );
        newp.setGain( oldp.getGain() );
      }

      FCSHandler  lmi = nfcs.getOutputIterator();
      for (int  i = 0;  lmi.hasMoreEvents();  ++i)
      {
        for (int  j = 0;  j < originalData.length;  ++j)
          lmi.writeValue( originalData[j][i] );
        for (int  j = 0;  j < new_data.length;  ++j)
          lmi.writeValue( new_data[j][i] );
      }
      lmi.close();
    }
```

```
        catch (Exception ex)
        {
          ex.printStackTrace();
        }
      }
    }


  }
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;
import javax.swing.event.TableModelListener;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame
extends JFrame
{
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton jButton1 = new JButton();
  private JButton jButton2 = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private JLabel statusBar = new JLabel();
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrumTable = new JTable();
  private JMenuItem menuSpectrum = new JMenuItem();
  private JFileChooser  fileChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );
  private JMenuItem jMenuItem1 = new JMenuItem();
  private JMenuItem jMenuItem2 = new JMenuItem();
  private JButton jButton4 = new JButton();
  private JComboBox functionChoice = new JComboBox();
  private JLabel jLabel1 = new JLabel();

  //Construct the frame
  public ConverterFrame()
  {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
```

```
      jbInit();
      spectrumTable.setModel( modelParameters );
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }
  //Component initialization
  private void jbInit() throws Exception  {
    image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
    image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
    image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Logicle Data Converter");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
      }
    });
    jButton1.setIcon(image1);
    jButton1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    jButton1.setToolTipText("Open File");
    jButton2.setIcon(image2);
    jButton2.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        save_action(e);
      }
    });
    jButton2.setToolTipText("Close File");
    jButton3.setIcon(image3);
    jButton3.setToolTipText("Help");
    menuSpectrum.setIcon(new
  ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
    menuSpectrum.setText("Spectrum");
```

```
menuSpectrum.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    spectrum_action(e);
  }
});
jMenuItem1.setIcon(image1);
jMenuItem1.setText("Data");
jMenuItem1.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    data_action(e);
  }
});
jMenuItem2.setIcon(image2);
jMenuItem2.setText("Save");
jMenuItem2.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    save_action(e);
  }
});
functionChoice.setFont(new java.awt.Font("Dialog", 1, 14));
functionChoice.setMaximumSize(new Dimension(32767, 27));
functionChoice.setMinimumSize(new Dimension(138, 27));
functionChoice.setPreferredSize(new Dimension(142, 27));
functionChoice.setModel(functionModel);
functionChoice.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    functionChoice(e);
  }
});
jButton4.setMaximumSize(new Dimension(51, 27));
jButton4.setMinimumSize(new Dimension(51, 27));
jButton4.setIcon(new
ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
jButton4.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    spectrum_action(e);
  }
});
jLabel1.setPreferredSize(new Dimension(17, 17));
jToolBar.add(jButton4, null);
jToolBar.add(jButton1);
jToolBar.add(jButton2);
jToolBar.add(jButton3);
jToolBar.add(jLabel1, null);
jToolBar.add(functionChoice, null);
jMenuFile.add(menuSpectrum);
jMenuFile.add(jMenuItem1);
jMenuFile.add(jMenuItem2);
jMenuFile.addSeparator();
jMenuFile.add(jMenuFileExit);
jMenuHelp.add(jMenuHelpAbout);
jMenuBar1.add(jMenuFile);
jMenuBar1.add(jMenuHelp);
this.setJMenuBar(jMenuBar1);
contentPane.add(jToolBar,  BorderLayout.NORTH);
```

```
  contentPane.add(statusBar, BorderLayout.SOUTH);
  contentPane.add(jScrollPane1, BorderLayout.CENTER);
  jScrollPane1.getViewport().add(spectrumTable, null);
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
}
//Help | About action performed
public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
}
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
  }
}

private static final String[]  stringArray = new String[0];
private static final int  N_PARAMETERS = 2;
private FCSFile  fcs;
private String[]  sensorNames = new String[0];
private int[][]  originalData;
private int[][]  newData;
private double[][]  rawData;
private double[][]  compensatedData;
private double[][]  compensationMatrix;

private class  LogicleFunction
{
  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double[]  lookup = new double[ parameter.getRange() ];
    lookup[0] = parameter.getMinimum();
    interpolate( lookup, 0, lookup.length, parameter.getMaximum() );

    return  lookup;
  }
}

private  LogicleFunction  logLinearSplice = new LogicleFunction ()
{

  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    if (Double.isNaN(modelParameters.getParameter( sensorNames.length, column )))
      return  super.transform( column, parameter );

    double[]  lookup = new double[ parameter.getRange() ];
    lookup[0] = parameter.getMinimum();
    interpolate( lookup, 0, lookup.length, parameter.getMaximum() );
    double  parameter1 = modelParameters.getParameter( sensorNames.length, column
);
    int  splice = (int) Math.round(
        (Math.log( parameter1 ) - Math.log( parameter.getMinimum() )) * (double)
parameter.getRange() / parameter.getDecades() / ln_10 );
    if (splice > 0 && splice < parameter.getRange() - 1)
    {
```

```
      double  delta = (lookup[splice+1] - lookup[splice-1])/2;
      double  base = lookup[splice];
      for (int  j = 0;  j < splice;  ++j)
        lookup[j] = base - (splice - j) * delta;
    }

    return  lookup;
  }
};
private LogicleFunction[]  transformer = { logLinearSplice };

private class  ModelParameters
extends AbstractTableModel
{
  private double[][]  parameters = new double[0][];

  public int getRowCount()
  {
    return  parameters.length;
  }

  public int getColumnCount()
  {
    return  sensorNames.length;
  }

  public String getColumnName (
      int columnIndex )
  {
    return  sensorNames[ columnIndex ];
  }

  public Class getColumnClass (
      int columnIndex )
  {
    return  super.getColumnClass( columnIndex );
  }

  public boolean isCellEditable (
      int rowIndex,
      int columnIndex )
  {
    return  true;
  }

  public Object getValueAt (
      int rowIndex,
      int columnIndex )
  {
    double  value = parameters[ rowIndex ][ columnIndex ];
    if (Double.isNaN( value ))
      return  null;
    else
      return  String.valueOf( value );
  }

  public void setValueAt (
      Object aValue,
      int rowIndex,
      int columnIndex)
  {
    double  dvalue;
    if (aValue instanceof Number)
```

```
      dvalue = ((Number) aValue).doubleValue();
    else if (aValue instanceof String)
    {
      try
      {
        dvalue = Double.valueOf( (String) aValue ).doubleValue();
      }
      catch (NumberFormatException ex)
      {
        dvalue = Double.NaN;
      }
    }
    else
      dvalue = Double.NaN;
    if (Double.isNaN( dvalue) && rowIndex < sensorNames.length)
      if ( rowIndex == columnIndex)
        dvalue = 1;
      else
        dvalue = 0;

    parameters[ rowIndex ][ columnIndex ] = dvalue;
    super.setValueAt( aValue, rowIndex, rowIndex );
  }

  public void  setParameters (
      double[][]  parameters )
  {
    this.parameters = parameters;
    this.fireTableStructureChanged();
  }

  public double  getParameter (
      int rowIndex,
      int columnIndex )
  {
    return  parameters[rowIndex][columnIndex];
  }
};

private ModelParameters  modelParameters = new ModelParameters();
private String[]  functions = { "log linear splice", "x + a log x + b",
"hyperbolic" };
private DefaultComboBoxModel functionModel = new DefaultComboBoxModel( functions
);
private static final double  ln_10 = Math.log( 10 );

private void  invert (
    double[][]  matrix )
{
  int  row = 0, col = 0, n = matrix.length;
  int  pivot[] = new int[n];
  int  row_index[] = new int[n];
  int  col_index[] = new int[n];

  for (int  i = 0;  i < n;  ++i)
  {
    double  big = 0;
    for (int  j = 0;  j < n;  ++j)
    {
      if (pivot[j] != 1)
        for (int  k = 0;  k < n;  ++k)
        {
          if (pivot[k] == 0)
```

```
        {
          double  abs = Math.abs(matrix[j][k]);
          if (abs >= big)
          {
            big = abs;
            row = j;
            col = k;
          }
        }
        else if (pivot[k] > 1)
          throw  new IllegalArgumentException( "matrix is singular" );
      }
    }
    ++pivot[col];
    row_index[i] = row;
    col_index[i] = col;

    if (row != col)
    {
      System.out.println( "pivoting!" );
      for (int  k = 0;  k < n;  ++k)
      {
        double  t = matrix[row][k];
        matrix[row][k] = matrix[col][k];
        matrix[col][k] = t;
      }
    }

    if (matrix[col][col] == 0)
      throw  new IllegalArgumentException( "matrix is singular" );
    double inverse = 1/matrix[col][col];
    matrix[col][col] = 1;
    for (int  j = 0;  j < n;  ++j)
      matrix[col][j] *= inverse;
    for (int  j = 0;  j < n;  ++j)
      if (j != col)
      {
        double  t = matrix[j][col];
        matrix[j][col] = 0;
        for (int  k = 0;  k < n;  ++k)
          matrix[j][k] -= matrix[col][k] * t;
      }
    }

    for (int  i = n-1;  i >= 0;  --i)
      if (row_index[i] != col_index[i])
        for (int  j = 0;  j < n;  ++j)
        {
          double  t = matrix[j][row_index[i]];
          matrix[j][row_index[i]] = matrix[j][col_index[i]];
          matrix[j][col_index[i]] = t;
        }
  }

  void  compensate ()
  throws IOException
  {
    int  n = fcs.getTotal();
    int  m = sensorNames.length;

    rawData = new double[m][n];
    for (int  i = 0;  i < m;  ++i)
    {
```

```
      FCSParameter  p = fcs.getParameter( sensorNames[i] );
      int[]  input_data = originalData[p.getIndex()-1];
      double[]  output_data = rawData[i];
      if (p.isLog())
      {
        double[]  antilog = new double[ p.getRange() ];
        antilog[0] = p.getMinimum();
        interpolate( antilog, 0, p.getRange(), p.getMaximum() );
        for (int  j = 0;  j < n;  ++j)
          output_data[j] = antilog[ input_data[j] ];
      }
      else
      {
        double  g = p.getGain();
        for (int  j = 0;  j < n;  ++i)
          output_data[j] = input_data[j] / g;
      }
    }

    double[][]  compensationMatrix = new double[m][m];
    for (int  i = 0;  i < m; ++i)
      for (int  j = 0;  j < m;  ++j)
        compensationMatrix[j][i] = modelParameters.getParameter( i, j );
    invert( compensationMatrix );

    compensatedData = new double[m][n];
    for (int  i = 0;  i < m;  ++i)
    {
      double[]  output_data = compensatedData[i];
      for (int  j = 0;  j < m;  ++j)
      {
        double[]  input_data = rawData[j];
        double  mult = compensationMatrix[i][j];
        for (int  k = 0;  k < n;  ++k)
          output_data[k] += input_data[k] * mult;
      }
    }

    newData = new int[m][n];
    for (int  i = 0;  i < m;  ++i)
    {
      FCSParameter  p = fcs.getParameter( sensorNames[i] );
      double[]  input_data = compensatedData[i];
      int[]  transformed_data = newData[i];
      if (p.isLog())
      {
        double[]  lookup = new double[ p.getRange() ];
        lookup[0] = p.getMinimum();
        interpolate( lookup, 0, p.getRange(), p.getMaximum() );
        switch (functionChoice.getSelectedIndex())
        {
          case 0:
            lookup[0] = p.getMinimum();
            interpolate( lookup, 0, p.getRange(), p.getMaximum() );
            double  parameter1 = modelParameters.getParameter( m, i );
            if (Double.isNaN( parameter1 ))
              break;
            int  splice = (int) Math.round(
                (Math.log( parameter1 ) - Math.log( p.getMinimum() )) * (double)
p.getRange() / p.getDecades() / ln_10 );
            if (splice > 0 && splice < p.getRange() - 1)
            {
              double  delta = (lookup[splice+1] - lookup[splice-1])/2;
```

```
            double  base = lookup[splice];
            for (int  j = 0;  j < splice;  ++j)
              lookup[j] = base - (splice - j) * delta;
          }
          break;

        case 1:
          lookup[0] = p.getMinimum();
          interpolate( lookup, 0, p.getRange(), p.getMaximum() );
          parameter1 = modelParameters.getParameter( m, i );
          if (Double.isNaN( parameter1 ))
            break;
          double  parameter2 = modelParameters.getParameter( m + 1, i );
          if (Double.isNaN( parameter2 ))
              parameter2 = 0;
          parameter2 += parameter1 * Math.log( p.getMinimum() ) / ln_10;
          parameter1 *= p.getDecades() / p.getRange();
          for (int  j = 0;  j < lookup.length;  ++j)
            lookup[j] += parameter1 * j + parameter2;
          break;

        case 2:
          lookup[0] = 1;
          interpolate( lookup, 0, p.getRange(), Math.exp( p.getDecades() * ln_10 )
);
          parameter1 = modelParameters.getParameter( m, i );
          if (Double.isNaN( parameter1 ))
            parameter1 = 0;
          double  x = p.getMinimum();
          for (int  j = 0;  j < lookup.length;  ++j)
            lookup[j] = x * (lookup[j] - 1/lookup[j]) - parameter1;
          break;
        }
        for (int  j = 0;  j < n;  ++j)
        {
          int  k = Arrays.binarySearch( lookup, input_data[j] );
          if (k < 0)
            transformed_data[j] = -k;
          else
            transformed_data[j] = k;
        }
      }
      else
      {
        double  g = p.getGain();
        for (int  j = 0;  j < n;  ++i)
          transformed_data[j] = (int) Math.round( input_data[j] * g );
      }
    }
  }

  void spectrum_action (
      ActionEvent e )
  {
    fileChooser.setDialogTitle( "Load Spectrum File" );
    int returnVal = fileChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        BufferedReader  br = new BufferedReader( new FileReader(
fileChooser.getSelectedFile() ) );
        br.readLine();
```

```
        br.readLine();
        StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
        ArrayList  al = new ArrayList();
        while (st.hasMoreTokens())
          al.add( st.nextToken() );
        sensorNames = (String[]) al.toArray( stringArray );

        double[][]  parameters = new double[sensorNames.length +
2][sensorNames.length];
        int  r;
        for (r = 0;  r < sensorNames.length;  ++r)
        {
          st = new StringTokenizer( br.readLine(), "\t" );
          for (int  c = 0;  c < sensorNames.length;  ++c)
            parameters[r][c] =  Double.parseDouble( st.nextToken() );
        }

        for (;  r < parameters.length;  ++r)
          for (int  c = 0;  c < sensorNames.length;  ++c)
            parameters[r][c] =  Double.NaN;

        modelParameters.setParameters( parameters );
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }

  private void  interpolate (
      double[]  func,
      int  i,
      int  n,
      double  x )
  {
    int  m = n / 2;
    double  y = Math.sqrt( func[i] * x );

    func[ i + m ] = y;

    if (m > 1 )
    {
      interpolate( func, i, m, y );
      interpolate( func, i + m, m, x );
    }

  }

  void data_action(ActionEvent e)
  {
    fileChooser.setDialogTitle( "Load FCS Data File" );
    int returnVal = fileChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        File  f = fileChooser.getSelectedFile();
        FCSFile  fcs = new FCSFile( f );
        int  n = fcs.getTotal();
        int  m = fcs.getParameters();
        int[][]  int_value = new int[m][n];
        FCSHandler  lmi = fcs.getInputIterator();
```

```
      for (int  i = 0;  lmi.hasMoreEvents();  ++i)
        for (int  j = 0;  lmi.hasMoreValues();  ++j)
          int_value[j][i] = lmi.readValue();
      this.originalData = int_value;
      this.fcs = fcs;
      statusBar.setText( f.getName() );
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
}

void save_action(ActionEvent e)
{
  fileChooser.setDialogTitle( "Save New FCS Data File" );
  int returnVal = fileChooser.showSaveDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      File  f = fileChooser.getSelectedFile();

      compensate();

      FCSFile  nfcs = new FCSFile( f );
      nfcs.getTextSegment().readFrom( fcs.getTextSegment() );
      nfcs.setFile( f );
      for (int  i = 0;  i < sensorNames.length;  ++i)
      {
        FCSParameter  oldp = fcs.getParameter( sensorNames[i] );
        FCSParameter  newp = nfcs.addParameter();
        newp.setAttribute( "$P", "N", "["+sensorNames[i]+"]" );
        newp.setAttribute( "$P", "S", oldp.getAttribute( "$P", "S" ) );
        newp.setBits( oldp.getBits() );
        newp.setRange( oldp.getRange() );
        newp.setLog( oldp.isLog() );
        newp.setDecades( oldp.getDecades() );
        newp.setMinimum( oldp.getMinimum() );
        newp.setGain( oldp.getGain() );

        double[]  input_data = compensatedData[i];
        int[]  transformed_data = newData[i];
        if (newp.isLog())
        {
          double[]  lookup = transformer[ functionChoice.getSelectedIndex()
].transform( i, newp );
          for (int  j = 0, n = fcs.getTotal();  j < n;  ++j)
          {
            int  k = Arrays.binarySearch( lookup, input_data[j] );
            if (k < 0)
              transformed_data[j] = -k-1;
            else
              transformed_data[j] = k;
          }
        }
        else
        {
          double  g = newp.getGain();
          for (int  j = 0, n = fcs.getTotal();  j < n;  ++i)
            transformed_data[j] = (int) Math.round( input_data[j] * g );
```

```
        }
      }

      FCSHandler  lmi = nfcs.getOutputIterator();
      for (int  i = 0;  lmi.hasMoreEvents();  ++i)
      {
        for (int  j = 0;  j < originalData.length;  ++j)
          lmi.writeValue( originalData[j][i] );
        for (int  j = 0;  j < newData.length;  ++j)
          lmi.writeValue( newData[j][i] );
      }
      lmi.close();
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
}

void functionChoice(ActionEvent e)
{
  for (int  i = 0;  i < sensorNames.length;  ++i)
    for (int  j = 0;  j < this.N_PARAMETERS;  ++j)
      modelParameters.setValueAt( null, sensorNames.length + i, j );
}


}
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;
import javax.swing.event.TableModelListener;
import com.borland.jbcl.layout.*;
import javax.swing.border.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame
extends JFrame
{
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton openButton = new JButton();
  private JButton saveButton = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrumTable = new JTable();
  private JMenuItem menuSpectrum = new JMenuItem();
  private JFileChooser  fileChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );
  private JMenuItem openMenuItem = new JMenuItem();
  private JMenuItem saveMenuItem = new JMenuItem();
  private JButton jButton4 = new JButton();
  private JComboBox functionChoice = new JComboBox();
  private JLabel jLabel1 = new JLabel();
  private JDialog helpDialog = new JDialog();
  private JPanel jPanel1 = new JPanel();
  private JButton helpDismiss = new JButton();
  private JScrollPane jScrollPane2 = new JScrollPane();
  private JEditorPane helpPane = new JEditorPane();
  private JPanel aboutMessage = new JPanel();
```

```
private JLabel jLabel2 = new JLabel();
private JLabel jLabel3 = new JLabel();
private GridLayout gridLayout1 = new GridLayout();
private JLabel jLabel4 = new JLabel();
private Component component1;
private Component component2;
private JLabel jLabel5 = new JLabel();
private JLabel jLabel6 = new JLabel();
private Border border1;
private Component component3;
private JLabel jLabel7 = new JLabel();
private JPanel jPanel2 = new JPanel();
private JLabel statusBar = new JLabel();
private JProgressBar progressBar = new JProgressBar();
private BorderLayout borderLayout3 = new BorderLayout();

//Construct the frame
public ConverterFrame()
{
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try
  {
    jbInit();
    spectrumTable.setModel( modelParameters );
    spectrumTable.getTableHeader().setReorderingAllowed( false );
    helpPane.setPage( ConverterFrame.class.getResource( "help.html" ) );
    helpDialog.pack();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
 }
 //Component initialization
 private void jbInit() throws Exception  {
   image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
    image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
    image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    component1 = Box.createVerticalStrut(8);
    component2 = Box.createVerticalStrut(8);
    border1 = BorderFactory.createEmptyBorder(10,15,10,25);
    component3 = Box.createVerticalStrut(8);
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Logicle Fluorescence Compensation");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About ...");
```

```
jMenuHelpAbout.addActionListener(new ActionListener()  {
  public void actionPerformed(ActionEvent e) {
    about_action(e);
  }
});
openButton.setIcon(image1);
openButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    data_action(e);
  }
});
openButton.setEnabled(false);
openButton.setToolTipText("Open FCS File");
saveButton.setIcon(image2);
saveButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    save_action(e);
  }
});
saveButton.setEnabled(false);
saveButton.setToolTipText("Save Extended FCS File");
jButton3.setIcon(image3);
jButton3.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    help_action(e);
  }
});
jButton3.setToolTipText("Help");
menuSpectrum.setIcon(new
ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
menuSpectrum.setText("Spectrum ...");
menuSpectrum.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    spectrum_action(e);
  }
});
openMenuItem.setEnabled(false);
openMenuItem.setIcon(image1);
openMenuItem.setText("Data ...");
openMenuItem.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    data_action(e);
  }
});
saveMenuItem.setEnabled(false);
saveMenuItem.setIcon(image2);
saveMenuItem.setText("Save As ...");
saveMenuItem.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    save_action(e);
  }
});
functionChoice.setFont(new java.awt.Font("Dialog", 1, 14));
```

```
    functionChoice.setMaximumSize(new Dimension(32767, 27));
    functionChoice.setMinimumSize(new Dimension(138, 27));
    functionChoice.setPreferredSize(new Dimension(142, 27));
    functionChoice.setToolTipText("Select a Data Transformation");
    functionChoice.setModel(functionModel);
    functionChoice.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        function_action(e);
      }
    });
    jButton4.setMaximumSize(new Dimension(51, 27));
    jButton4.setMinimumSize(new Dimension(51, 27));
    jButton4.setToolTipText("Load Spectrum Matrix");
    jButton4.setIcon(new
ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
    jButton4.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        spectrum_action(e);
      }
    });
    jLabel1.setPreferredSize(new Dimension(17, 17));
    spectrumTable.setRowSelectionAllowed(false);
    helpDismiss.setText("Close");
    helpDismiss.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        help_dismiss(e);
      }
    });
    helpDialog.setTitle("Logicle Help");
    helpDialog.getContentPane().setLayout(borderLayout2);
    jMenuItem2.setText("Help ...");
    jMenuItem2.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        help_action(e);
      }
    });
    helpPane.setPreferredSize(new Dimension(400, 400));
    helpPane.setEditable(false);
    helpPane.setText("<html>\r\n  <head>\r\n\r\n  </head>\r\n  <body>\r\n    <p>\r\n
Logicle Data Converter " +
    "    \r\n    </p>\r\n  </body>\r\n</html>\r\n");
    helpPane.setContentType("text/html");
    jScrollPane2.setPreferredSize(new Dimension(400, 400));
    jLabel2.setFont(new java.awt.Font("Dialog", 1, 14));
    jLabel2.setAlignmentY((float) 0.0);
    jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel2.setHorizontalTextPosition(SwingConstants.LEADING);
    jLabel2.setIconTextGap(0);
    jLabel2.setText("Logicle Fluorescence Compensation");
    jLabel3.setAlignmentX((float) 0.5);
    jLabel3.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel3.setIconTextGap(0);
    jLabel3.setText("Wayne A. Moore");
    aboutMessage.setLayout(gridLayout1);
    gridLayout1.setColumns(1);
```

```
gridLayout1.setRows(9);
jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
jLabel4.setIconTextGap(0);
jLabel4.setText("August 2002");
jLabel5.setFont(new java.awt.Font("Dialog", 2, 10));
jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
jLabel5.setHorizontalTextPosition(SwingConstants.LEADING);
jLabel5.setIconTextGap(0);
jLabel5.setText("Copyright 2002, by The Board of Trustees");
jLabel6.setFont(new java.awt.Font("Dialog", 2, 10));
jLabel6.setHorizontalAlignment(SwingConstants.CENTER);
jLabel6.setHorizontalTextPosition(SwingConstants.LEADING);
jLabel6.setIconTextGap(0);
jLabel6.setText("of the Leland Stanford Jr. University");
aboutMessage.setBorder(border1);
aboutMessage.setMinimumSize(new Dimension(60, 40));
aboutMessage.setPreferredSize(new Dimension(300, 200));
jLabel7.setHorizontalAlignment(SwingConstants.CENTER);
jLabel7.setText("David R. Parks");
statusBar.setText(" ");
jPanel2.setLayout(borderLayout3);
progressBar.setString("Compensate");
jToolBar.add(jButton4, null);
jToolBar.add(openButton);
jToolBar.add(saveButton);
jToolBar.add(jButton3);
jToolBar.add(jLabel1, null);
jToolBar.add(functionChoice, null);
jMenuFile.add(menuSpectrum);
jMenuFile.add(openMenuItem);
jMenuFile.add(saveMenuItem);
jMenuFile.addSeparator();
jMenuFile.add(jMenuFileExit);
jMenuHelp.add(jMenuItem2);
jMenuHelp.addSeparator();
jMenuHelp.add(jMenuHelpAbout);
jMenuBar1.add(jMenuFile);
jMenuBar1.add(jMenuHelp);
this.setJMenuBar(jMenuBar1);
contentPane.add(jToolBar,  BorderLayout.NORTH);
contentPane.add(jScrollPane1, BorderLayout.CENTER);
contentPane.add(jPanel2, BorderLayout.SOUTH);
jPanel2.add(statusBar,  BorderLayout.CENTER);
jPanel2.add(progressBar,  BorderLayout.EAST);
jScrollPane1.getViewport().add(spectrumTable, null);
helpDialog.getContentPane().add(jPanel1, BorderLayout.SOUTH);
jPanel1.add(helpDismiss, null);
helpDialog.getContentPane().add(jScrollPane2, BorderLayout.CENTER);
jScrollPane2.getViewport().add(helpPane, null);
aboutMessage.add(jLabel2, null);
aboutMessage.add(component1, null);
aboutMessage.add(jLabel3, null);
aboutMessage.add(jLabel7, null);
aboutMessage.add(jLabel4, null);
aboutMessage.add(component2, null);
aboutMessage.add(jLabel5, null);
aboutMessage.add(jLabel6, null);
aboutMessage.add(component3, null);
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
}
```

```
//Help | About action performed
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
  }
}

private static final String[]  stringArray = new String[0];
private static final int  N_PARAMETERS = 2;
private FCSFile  fcs;
private String fcs_name;
private String[]  sensorNames = new String[0];
private int[][]  originalData;
private int[][]  newData;
private double[][]  rawData;
private double[][]  compensatedData;
private double[][]  compensationMatrix;
private int  progress;

public class  LogicleFunction
{
  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double[]  lookup = new double[ parameter.getRange() ];
    lookup[0] = parameter.getMinimum();
    interpolate( lookup, 0, lookup.length, parameter.getMaximum() );

    return  lookup;
  }
}

public final  LogicleFunction  logLinearSplice = new LogicleFunction ()
{

  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double  linear_range = modelParameters.getParameter( sensorNames.length,
column );
    if (Double.isNaN( linear_range ))
      return  super.transform( column, parameter );
    double ln_range = Math.log( parameter.getMaximum() / linear_range ) + 2;
    parameter.setDecades( ln_range / ln_10, parameter.getMaximum() / Math.exp(
ln_range ) );

    double[]  lookup = new double[ parameter.getRange() ];
    lookup[0] = parameter.getMinimum();
    interpolate( lookup, 0, lookup.length, parameter.getMaximum() );

    int  splice = (int) Math.round(
        Math.log( linear_range / parameter.getMinimum() ) * (double)
parameter.getRange() / parameter.getDecades() / ln_10 );
    if (splice > 0 && splice < parameter.getRange() - 1)
    {
      double  delta = (lookup[splice+1] - lookup[splice-1])/2;
      double  base = lookup[splice];
      for (int  j = 0;  j < splice;  ++j)
        lookup[j] = base - (splice - j) * delta;
```

```
    }

    return  lookup;
  }
};

public final  LogicleFunction  hyperbolic = new LogicleFunction ()
{
  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double  linear_range = modelParameters.getParameter( sensorNames.length,
column );
    if (Double.isNaN( linear_range ))
      return  super.transform( column, parameter );
    double  scale_factor = linear_range / (Math.E - 1/Math.E);
    double ln_range = Math.log( parameter.getMaximum() / scale_factor ) + 2;
    parameter.setDecades( ln_range / ln_10, parameter.getMaximum() / Math.exp(
ln_range ) );

    double[]  lookup = new double[ parameter.getRange() ];
    lookup[0] = 1 / Math.E;
    interpolate( lookup, 0, lookup.length, parameter.getMaximum() / scale_factor
);

    for (int  j = 0;  j < lookup.length;  ++j)
      lookup[j] = (lookup[j] - 1/lookup[j]) * scale_factor;

    return  lookup;
  }
};

public final  LogicleFunction  logPlus = new LogicleFunction ()
{
  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double  parameter1 = modelParameters.getParameter( sensorNames.length, column
);
    double  parameter2 = modelParameters.getParameter( sensorNames.length + 1,
column );
    if (Double.isNaN( parameter1 ))
      return  super.transform( column, parameter );
    if (Double.isNaN( parameter2 ))
      parameter2 = 0;

    double[]  lookup = new double[ parameter.getRange() ];
    lookup[0] = parameter.getMinimum();
    interpolate( lookup, 0, lookup.length, parameter.getMaximum() );

//    parameter2 += parameter1 * Math.log( parameter.getMinimum() ) / ln_10;
    parameter1 *= parameter.getDecades() / parameter.getRange();
    for (int  j = 0;  j < lookup.length;  ++j)
      lookup[j] += parameter1 * j + parameter2;

    return  lookup;
  }
};

private class  ModelParameters
```

```
extends AbstractTableModel
{
  private double[][]  parameters = new double[0][];

  public int getRowCount()
  {
    return  parameters.length;
  }

  public int getColumnCount()
  {
    return  sensorNames.length;
  }

  public String getColumnName (
      int columnIndex )
  {
    return  sensorNames[ columnIndex ];
  }

  public Class getColumnClass (
      int columnIndex )
  {
    return  super.getColumnClass( columnIndex );
  }

  public boolean isCellEditable (
      int rowIndex,
      int columnIndex )
  {
    return  true;
  }

  public Object getValueAt (
      int rowIndex,
      int columnIndex )
  {
    double  value = parameters[ rowIndex ][ columnIndex ];
    if (Double.isNaN( value ))
      return  null;
    else
      return  String.valueOf( value );
  }

  public void setValueAt (
      Object aValue,
      int rowIndex,
      int columnIndex)
  {
    double  dvalue;
    if (aValue instanceof Number)
      dvalue = ((Number) aValue).doubleValue();
    else if (aValue instanceof String)
    {
      try
      {
        dvalue = Double.valueOf( (String) aValue ).doubleValue();
      }
      catch (NumberFormatException ex)
      {
        dvalue = Double.NaN;
      }
    }
```

```
      else
        dvalue = Double.NaN;
      if (Double.isNaN( dvalue) && rowIndex < sensorNames.length)
        if ( rowIndex == columnIndex)
          dvalue = 1;
        else
          dvalue = 0;

      parameters[ rowIndex ][ columnIndex ] = dvalue;
      fireTableCellUpdated( rowIndex, columnIndex );
    }

    public void  setParameters (
        double[][]  parameters )
    {
      this.parameters = parameters;
      this.fireTableStructureChanged();
    }

    public double  getParameter (
        int rowIndex,
        int columnIndex )
    {
      return  parameters[rowIndex][columnIndex];
    }
  };

  private ModelParameters  modelParameters = new ModelParameters();
  private String[]  functionNames =
  { "log linear splice", "hyperbolic", "x + a log x + b" };
  private LogicleFunction[]  logicleFunctions =
  { logLinearSplice,     hyperbolic,   logPlus };
  private DefaultComboBoxModel functionModel = new DefaultComboBoxModel(
functionNames );
  private static final double  ln_10 = Math.log( 10 );
  private BorderLayout borderLayout2 = new BorderLayout();
  private JMenuItem jMenuItem2 = new JMenuItem();
  private final String  SPECTRUM_ERROR = "The data are not compatible with the
spectrum.";

  private void  interpolate (
      double[]  func,
      int  i,
      int  n,
      double  x )
  {
    int  m = n / 2;
    double  y = Math.sqrt( func[i] * x );

    func[ i + m ] = y;

    if (m > 1 )
    {
      interpolate( func, i, m, y );
      interpolate( func, i + m, m, x );
    }

  }

  public void  invert (
      double[][]  matrix )
  {
    int  row = 0, col = 0, n = matrix.length;
```

```
int  pivot[] = new int[n];
int  row_index[] = new int[n];
int  col_index[] = new int[n];

for (int  i = 0;  i < n;  ++i)
{
  double  big = 0;
  for (int  j = 0;  j < n;  ++j)
  {
    if (pivot[j] != 1)
      for (int  k = 0;  k < n;  ++k)
      {
        if (pivot[k] == 0)
        {
          double  abs = Math.abs(matrix[j][k]);
          if (abs >= big)
          {
            big = abs;
            row = j;
            col = k;
          }
        }
        else if (pivot[k] > 1)
          throw  new IllegalArgumentException( "matrix is singular" );
      }
  }
  ++pivot[col];
  row_index[i] = row;
  col_index[i] = col;

  if (row != col)
    for (int  k = 0;  k < n;  ++k)
    {
      double  t = matrix[row][k];
      matrix[row][k] = matrix[col][k];
      matrix[col][k] = t;
    }

  if (matrix[col][col] == 0)
    throw  new IllegalArgumentException( "matrix is singular" );
  double inverse = 1/matrix[col][col];
  matrix[col][col] = 1;
  for (int  j = 0;  j < n;  ++j)
    matrix[col][j] *= inverse;
  for (int  j = 0;  j < n;  ++j)
    if (j != col)
    {
      double  t = matrix[j][col];
      matrix[j][col] = 0;
      for (int  k = 0;  k < n;  ++k)
        matrix[j][k] -= matrix[col][k] * t;
    }
}

for (int  i = n-1;  i >= 0;  --i)
  if (row_index[i] != col_index[i])
    for (int  j = 0;  j < n;  ++j)
    {
      double  t = matrix[j][row_index[i]];
      matrix[j][row_index[i]] = matrix[j][col_index[i]];
      matrix[j][col_index[i]] = t;
    }
}
```

```
void  compensate ()
throws IOException
{
  int  n = fcs.getTotal();
  int  m = sensorNames.length;

  rawData = new double[m][n];
  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensorNames[i] );
    int[]  input_data = originalData[p.getIndex()-1];
    double[]  output_data = rawData[i];
    if (p.isLog())
    {
      double[]  antilog = new double[ p.getRange() ];
      antilog[0] = p.getMinimum();
      interpolate( antilog, 0, p.getRange(), p.getMaximum() );
      for (int  j = 0;  j < n;  ++j)
        output_data[j] = antilog[ input_data[j] ];
    }
    else
    {
      double  g = p.getGain();
      for (int  j = 0;  j < n;  ++j)
        output_data[j] = input_data[j] / g;
    }
    progressBar.setValue( ++progress );
  }

  double[][]  compensationMatrix = new double[m][m];
  for (int  i = 0;  i < m; ++i)
    for (int  j = 0;  j < m;  ++j)
      compensationMatrix[j][i] = modelParameters.getParameter( i, j );
  invert( compensationMatrix );

  compensatedData = new double[m][n];
  for (int  i = 0;  i < m;  ++i)
  {
    double[]  output_data = compensatedData[i];
    for (int  j = 0;  j < m;  ++j)
    {
      double[]  input_data = rawData[j];
      double  coefficient = compensationMatrix[i][j];
      for (int  k = 0;  k < n;  ++k)
        output_data[k] += input_data[k] * coefficient;
    }
    progressBar.setValue( ++progress );
  }
}

void spectrum_action (
    ActionEvent e )
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();

  fileChooser.setDialogTitle( "Load Spectrum File" );
  int returnVal = fileChooser.showOpenDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
```

```
        BufferedReader  br = new BufferedReader( new FileReader(
fileChooser.getSelectedFile() ) );
        br.readLine();
        br.readLine();
        StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
        ArrayList  al = new ArrayList();
        while (st.hasMoreTokens())
          al.add( st.nextToken() );
        sensorNames = (String[]) al.toArray( stringArray );

        double[][]  parameters = new double[sensorNames.length +
2][sensorNames.length];
        int  r;
        for (r = 0;  r < sensorNames.length;  ++r)
        {
          st = new StringTokenizer( br.readLine(), "\t" );
          for (int  c = 0;  c < sensorNames.length;  ++c)
            parameters[r][c] =  Double.parseDouble( st.nextToken() );
        }

        for (;  r < parameters.length;  ++r)
          for (int  c = 0;  c < sensorNames.length;  ++c)
            parameters[r][c] =  Double.NaN;

        modelParameters.setParameters( parameters );
        openButton.setEnabled( true );
        openMenuItem.setEnabled( true );
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }

  void data_action(ActionEvent e)
  {
    if (spectrumTable.isEditing())
      spectrumTable.getCellEditor().stopCellEditing();

    fileChooser.setDialogTitle( "Load FCS Data File" );
    int returnVal = fileChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        File  f = fileChooser.getSelectedFile();
        statusBar.setText( f.getName() );
        progress = 0;
        progressBar.setMinimum( 0 );
        progressBar.setValue( 0 );

        FCSFile  fcs = new FCSFile( f );
        for (int  i = 0;  i < sensorNames.length;  ++i)
          if (fcs.getParameter( sensorNames[i] ) == null)
          {
            JOptionPane.showMessageDialog( this, SPECTRUM_ERROR,
                "Error", JOptionPane.ERROR_MESSAGE );
            return;
          }

        int  n = fcs.getTotal();
        progressBar.setMaximum( n );
```

```
        progressBar.setEnabled( true );
        int  m = fcs.getParameters();
        int[][]  int_value = new int[m][n];
        FCSHandler  lmi = fcs.getInputIterator();
        for (int  i = 0;  lmi.hasMoreEvents();  ++i)
        {
          for (int  j = 0;  lmi.hasMoreValues();  ++j)
            int_value[j][i] = lmi.readValue();
          progressBar.setValue( ++progress );
        }
        this.originalData = int_value;
        this.fcs = fcs;
        fcs_name = f.getName();

        statusBar.setText( fcs_name );
        progressBar.setValue( 0 );
        progressBar.setEnabled( false );
        saveButton.setEnabled( true );
        saveMenuItem.setEnabled( true );
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }

  void save_action(ActionEvent e)
  {
    if (spectrumTable.isEditing())
      spectrumTable.getCellEditor().stopCellEditing();

    fileChooser.setDialogTitle( "Save New FCS Data File" );
    int returnVal = fileChooser.showSaveDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        File  f = fileChooser.getSelectedFile();
        LogicleFunction  logicle = logicleFunctions[
functionChoice.getSelectedIndex() ];

        if (f.exists())
        {
          Object[]  message = { f, "already exists. Do you want to replace it?" };
          if (JOptionPane.OK_OPTION !=
              JOptionPane.showConfirmDialog( this, message, "Overwrite",
              JOptionPane.OK_CANCEL_OPTION, JOptionPane.WARNING_MESSAGE ))
            return;
        }

        progress = 0;
        int  m = sensorNames.length;
        progressBar.setMinimum( 0 );
        progressBar.setMaximum( 2 * m + m * m  );
        progressBar.setValue( 0 );
        progressBar.setEnabled( true );
        statusBar.setText( "Working" );

        rawData = null;
        compensatedData = null;
        newData = null;
        try
```

```
        {
          compensate();
        }
        catch (IllegalArgumentException ex)
        {
          JOptionPane.showMessageDialog( this, "Spectrum Matrix is Singular",
"Error",
              JOptionPane.ERROR_MESSAGE );
          return;
        }
        rawData = null;

        FCSFile  newFcs = new FCSFile(  fileChooser.getSelectedFile() );
        newData = new int[sensorNames.length][fcs.getTotal()];
        newFcs.getTextSegment().readFrom( fcs.getTextSegment() );
        for (int  i = 0;  i < sensorNames.length;  ++i)
        {
          FCSParameter  oldp = fcs.getParameter( sensorNames[i] );
          if (oldp == null)
          {
            JOptionPane.showMessageDialog( this, SPECTRUM_ERROR,
               "Error", JOptionPane.ERROR_MESSAGE );
            return;
          }
          FCSParameter  newp = newFcs.addParameter();
          newp.setAttribute( "$P", "N", "["+sensorNames[i]+"]" );
          newp.setAttribute( "$P", "S", oldp.getAttribute( "$P", "S" ) );
          newp.setBits( oldp.getBits() );
          newp.setRange( oldp.getRange() );
          newp.setDecades( oldp.getDecades(), oldp.getMinimum() );
          newp.setGain( oldp.getGain() );

          double[]  input_data = compensatedData[i];
          int[]  transformed_data = newData[i];
          if (newp.isLog())
          {
            double[]  lookup = logicle.transform( i, newp );
            for (int  j = 0, n = fcs.getTotal();  j < n;  ++j)
            {
              int  k = Arrays.binarySearch( lookup, input_data[j] );
              if (k < 0)
                transformed_data[j] = -k-1;
              else
                transformed_data[j] = k;
            }
          }
          else
          {
            double  g = newp.getGain();
            for (int  j = 0, n = fcs.getTotal();  j < n;  ++i)
              transformed_data[j] = (int) Math.round( input_data[j] * g );
          }
          progressBar.setValue( ++progress );
        }
        compensatedData = null;

        progress = 0;
        progressBar.setValue( 0 );
        progressBar.setMaximum( fcs.getTotal() );
        statusBar.setText( f.getName() );

        FCSHandler  lmi = newFcs.getOutputIterator();
        for (int  i = 0;  lmi.hasMoreEvents();  ++i)
```

```
      {
        for (int  j = 0;  j < originalData.length;  ++j)
          lmi.writeValue( originalData[j][i] );
        for (int  j = 0;  j < newData.length;  ++j)
          lmi.writeValue( newData[j][i] );
        progressBar.setValue( ++progress );
      }
      lmi.close();
      newData = null;

      progressBar.setValue( 0 );
      progressBar.setEnabled( false );
      statusBar.setText( fcs_name );
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
}

void function_action(ActionEvent e)
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();

  for (int  i = 0;  i < sensorNames.length;  ++i)
    for (int  j = 0;  j < this.N_PARAMETERS;  ++j)
      modelParameters.setValueAt( null, sensorNames.length + j, i );
}

void help_action(ActionEvent e)
{
  helpDialog.setLocationRelativeTo( this );
  helpDialog.setVisible( true );
}

void help_dismiss(ActionEvent e)
{
  helpDialog.setVisible( false );
}

void about_action(ActionEvent e)
{
  JOptionPane.showMessageDialog( this, aboutMessage, "About Logicle",
JOptionPane.PLAIN_MESSAGE );
  }
}
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;
import javax.swing.event.TableModelListener;
import com.borland.jbcl.layout.*;
import javax.swing.border.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame
extends JFrame
{
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton openButton = new JButton();
  private JButton saveButton = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrumTable = new JTable();
  private JMenuItem spectrumMenuItem = new JMenuItem();
  private JFileChooser  fileChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );
  private JMenuItem openMenuItem = new JMenuItem();
  private JMenuItem saveMenuItem = new JMenuItem();
  private JButton spectrumButton = new JButton();
  private JComboBox functionChoice = new JComboBox();
  private JLabel jLabel1 = new JLabel();
  private JDialog helpDialog = new JDialog();
  private JPanel jPanel1 = new JPanel();
  private JButton helpDismiss = new JButton();
  private JScrollPane jScrollPane2 = new JScrollPane();
  private JEditorPane helpPane = new JEditorPane();
  private JPanel aboutMessage = new JPanel();
```

```
private JLabel jLabel2 = new JLabel();
private JLabel jLabel3 = new JLabel();
private GridLayout gridLayout1 = new GridLayout();
private JLabel jLabel4 = new JLabel();
private Component component2;
private JLabel jLabel5 = new JLabel();
private JLabel jLabel6 = new JLabel();
private Border border1;
private Component component3;
private JLabel jLabel7 = new JLabel();
private JPanel jPanel2 = new JPanel();
private JLabel statusBar = new JLabel();
private JProgressBar progressBar = new JProgressBar();
private BorderLayout borderLayout3 = new BorderLayout();
private BorderLayout borderLayout2 = new BorderLayout();
private JMenuItem jMenuItem2 = new JMenuItem();

//Construct the frame
public ConverterFrame()
{
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try
  {
    jbInit();
    spectrumTable.setModel( modelParameters );
    spectrumTable.getTableHeader().setReorderingAllowed( false );
    helpPane.setPage( ConverterFrame.class.getResource( "help.html" ) );
    helpDialog.pack();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
 }
 //Component initialization
 private void jbInit() throws Exception  {
   image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
   image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
   image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    component2 = Box.createVerticalStrut(8);
    border1 = BorderFactory.createEmptyBorder(10,15,10,25);
    component3 = Box.createVerticalStrut(8);
    component1 = Box.createVerticalStrut(8);
    component4 = Box.createVerticalStrut(8);
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Logicle Fluorescence Compensation");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
```

```
jMenuHelp.setText("Help");
jMenuHelpAbout.setText("About ...");
jMenuHelpAbout.addActionListener(new ActionListener()  {
  public void actionPerformed(ActionEvent e) {
    about_action(e);
  }
});
openButton.setIcon(image1);
openButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    data_action(e);
  }
});
openButton.setEnabled(false);
openButton.setToolTipText("Open FCS File");
saveButton.setIcon(image2);
saveButton.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    save_action(e);
  }
});
saveButton.setEnabled(false);
saveButton.setToolTipText("Save Extended FCS File");
jButton3.setIcon(image3);
jButton3.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    help_action(e);
  }
});
jButton3.setToolTipText("Help");
spectrumMenuItem.setIcon(new
ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
spectrumMenuItem.setText("Spectrum ...");
spectrumMenuItem.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    spectrum_action(e);
  }
});
openMenuItem.setEnabled(false);
openMenuItem.setIcon(image1);
openMenuItem.setText("Data ...");
openMenuItem.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    data_action(e);
  }
});
saveMenuItem.setEnabled(false);
saveMenuItem.setIcon(image2);
saveMenuItem.setText("Save As ...");
saveMenuItem.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    save_action(e);
  }
```

```
    });
    functionChoice.setFont(new java.awt.Font("Dialog", 1, 14));
    functionChoice.setMaximumSize(new Dimension(32767, 27));
    functionChoice.setMinimumSize(new Dimension(138, 27));
    functionChoice.setPreferredSize(new Dimension(142, 27));
    functionChoice.setToolTipText("Select a Data Transformation");
    functionChoice.setModel(functionModel);
    functionChoice.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        function_action(e);
      }
    });
    spectrumButton.setMaximumSize(new Dimension(51, 27));
    spectrumButton.setMinimumSize(new Dimension(51, 27));
    spectrumButton.setToolTipText("Load Spectrum Matrix");
    spectrumButton.setIcon(new
ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
    spectrumButton.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        spectrum_action(e);
      }
    });
    jLabel1.setPreferredSize(new Dimension(17, 17));
    spectrumTable.setRowSelectionAllowed(false);
    helpDismiss.setText("Close");
    helpDismiss.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        help_dismiss(e);
      }
    });
    helpDialog.setTitle("Logicle Help");
    helpDialog.getContentPane().setLayout(borderLayout2);
    jMenuItem2.setText("Help ...");
    jMenuItem2.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        help_action(e);
      }
    });
    helpPane.setPreferredSize(new Dimension(400, 400));
    helpPane.setEditable(false);
    helpPane.setText("<html>\r\n  <head>\r\n\r\n  </head>\r\n  <body>\r\n    <p>\r\n
Logicle Data Converter " +
    "    \r\n    </p>\r\n  </body>\r\n</html>\r\n");
    helpPane.setContentType("text/html");
    jScrollPane2.setPreferredSize(new Dimension(400, 400));
    jLabel2.setFont(new java.awt.Font("Dialog", 1, 14));
    jLabel2.setAlignmentY((float) 0.0);
    jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel2.setHorizontalTextPosition(SwingConstants.LEADING);
    jLabel2.setIconTextGap(0);
    jLabel2.setText("Logicle Fluorescence Compensation");
    jLabel3.setAlignmentX((float) 0.5);
    jLabel3.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel3.setIconTextGap(0);
    jLabel3.setText("Wayne A. Moore");
```

```
aboutMessage.setLayout(gridLayout1);
gridLayout1.setColumns(1);
gridLayout1.setRows(11);
jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
jLabel4.setIconTextGap(0);
jLabel4.setText("August 2002");
jLabel5.setFont(new java.awt.Font("Dialog", 2, 10));
jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
jLabel5.setHorizontalTextPosition(SwingConstants.LEADING);
jLabel5.setIconTextGap(0);
jLabel5.setText("Copyright 2002, by The Board of Trustees");
jLabel6.setFont(new java.awt.Font("Dialog", 2, 10));
jLabel6.setHorizontalAlignment(SwingConstants.CENTER);
jLabel6.setHorizontalTextPosition(SwingConstants.LEADING);
jLabel6.setIconTextGap(0);
jLabel6.setText("of the Leland Stanford Jr. University");
aboutMessage.setBorder(border1);
aboutMessage.setMinimumSize(new Dimension(60, 40));
aboutMessage.setPreferredSize(new Dimension(300, 250));
jLabel7.setHorizontalAlignment(SwingConstants.CENTER);
jLabel7.setText("David R. Parks");
statusBar.setText(" ");
jPanel2.setLayout(borderLayout3);
progressBar.setString("Compensate");
jLabel8.setHorizontalAlignment(SwingConstants.CENTER);
jLabel8.setText("Version 0.3");
jToolBar.add(spectrumButton, null);
jToolBar.add(openButton);
jToolBar.add(saveButton);
jToolBar.add(jButton3);
jToolBar.add(jLabel1, null);
jToolBar.add(functionChoice, null);
jMenuFile.add(spectrumMenuItem);
jMenuFile.add(openMenuItem);
jMenuFile.add(saveMenuItem);
jMenuFile.addSeparator();
jMenuFile.add(jMenuFileExit);
jMenuHelp.add(jMenuItem2);
jMenuHelp.addSeparator();
jMenuHelp.add(jMenuHelpAbout);
jMenuBar1.add(jMenuFile);
jMenuBar1.add(jMenuHelp);
this.setJMenuBar(jMenuBar1);
contentPane.add(jToolBar,  BorderLayout.NORTH);
contentPane.add(jScrollPane1, BorderLayout.CENTER);
contentPane.add(jPanel2, BorderLayout.SOUTH);
jPanel2.add(statusBar,  BorderLayout.CENTER);
jPanel2.add(progressBar,  BorderLayout.EAST);
jScrollPane1.getViewport().add(spectrumTable, null);
helpDialog.getContentPane().add(jPanel1, BorderLayout.SOUTH);
jPanel1.add(helpDismiss, null);
helpDialog.getContentPane().add(jScrollPane2, BorderLayout.CENTER);
jScrollPane2.getViewport().add(helpPane, null);
aboutMessage.add(jLabel2, null);
aboutMessage.add(component4, null);
aboutMessage.add(jLabel8, null);
aboutMessage.add(component1, null);
aboutMessage.add(jLabel3, null);
aboutMessage.add(jLabel7, null);
aboutMessage.add(jLabel4, null);
aboutMessage.add(component2, null);
aboutMessage.add(jLabel5, null);
aboutMessage.add(jLabel6, null);
```

```
  aboutMessage.add(component3, null);
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
}
//Help | About action performed
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
  }
}

private static final String[]  stringArray = new String[0];
private static final int  N_PARAMETERS = 2;
private static final int FUZZY_BITS = 8;
private FCSFile  fcs;
private String[]  sensorNames = new String[0];
private int[][]  originalData;
private int[][]  newData;
private double[][]  rawData;
private double[][]  compensatedData;
private double[][]  compensationMatrix;
private int  progress;
protected LogicleFunction  logicle;

public class  LogicleFunction
{
  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double[]  lookup = new double[ parameter.getRange() ];
    lookup[0] = parameter.getMinimum();
    interpolate( lookup, 0, lookup.length, parameter.getMaximum() );

    return  lookup;
  }
}

public final  LogicleFunction  logLinearSplice = new LogicleFunction ()
{

  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double  linear_range = modelParameters.getParameter( sensorNames.length,
column );
    if (Double.isNaN( linear_range ))
      return  super.transform( column, parameter );
    double ln_range = Math.log( parameter.getMaximum() / linear_range ) + 2;
    parameter.setDecades( ln_range / ln_10, parameter.getMaximum() / Math.exp(
ln_range ) );

    double[]  lookup = new double[ parameter.getRange() ];
    lookup[0] = parameter.getMinimum();
    interpolate( lookup, 0, lookup.length, parameter.getMaximum() );

    int  splice = (int) Math.round(
        Math.log( linear_range / parameter.getMinimum() ) * (double)
```

```
parameter.getRange() / parameter.getDecades() / ln_10 );
      if (splice > 0 && splice < parameter.getRange() - 1)
      {
        double  delta = (lookup[splice+1] - lookup[splice-1])/2;
        double  base = lookup[splice];
        for (int  j = 0;  j < splice;  ++j)
          lookup[j] = base - (splice - j) * delta;
      }

      return  lookup;
    }
  };

  public final  LogicleFunction  hyperbolic = new LogicleFunction ()
  {

    public double[]  transform (
        int  column,
        FCSParameter  parameter )
    {
      double  linear_range = modelParameters.getParameter( sensorNames.length,
column );
      if (Double.isNaN( linear_range ))
        return  super.transform( column, parameter );
      double  scale_factor = linear_range / (Math.E - 1/Math.E);
      double ln_range = Math.log( parameter.getMaximum() / scale_factor ) + 2;
      parameter.setDecades( ln_range / ln_10, parameter.getMaximum() / Math.exp(
ln_range ) );

      double[]  lookup = new double[ parameter.getRange() ];
      lookup[0] = 1 / Math.E;
      interpolate( lookup, 0, lookup.length, parameter.getMaximum() / scale_factor
);

      for (int  j = 0;  j < lookup.length;  ++j)
        lookup[j] = (lookup[j] - 1/lookup[j]) * scale_factor;

      return  lookup;
    }
  };

  public final  LogicleFunction  logPlus = new LogicleFunction ()
  {
    public double[]  transform (
        int  column,
        FCSParameter  parameter )
    {
      double  parameter1 = modelParameters.getParameter( sensorNames.length, column
);
      double  parameter2 = modelParameters.getParameter( sensorNames.length + 1,
column );
      if (Double.isNaN( parameter1 ))
        return  super.transform( column, parameter );
      if (Double.isNaN( parameter2 ))
        parameter2 = 0;

      double[]  lookup = new double[ parameter.getRange() ];
      lookup[0] = parameter.getMinimum();
      interpolate( lookup, 0, lookup.length, parameter.getMaximum() );

//      parameter2 += parameter1 * Math.log( parameter.getMinimum() ) / ln_10;
      parameter1 *= parameter.getDecades() / parameter.getRange();
      for (int  j = 0;  j < lookup.length;  ++j)
```

```
      lookup[j] += parameter1 * j + parameter2;

    return  lookup;
  }
};

private class  ModelParameters
extends AbstractTableModel
{
  private double[][]  parameters = new double[0][];

  public int getRowCount()
  {
    return  parameters.length;
  }

  public int getColumnCount()
  {
    return  sensorNames.length;
  }

  public String getColumnName (
      int columnIndex )
  {
    return  sensorNames[ columnIndex ];
  }

  public Class getColumnClass (
      int columnIndex )
  {
    return  super.getColumnClass( columnIndex );
  }

  public boolean isCellEditable (
      int rowIndex,
      int columnIndex )
  {
    return  true;
  }

  public Object getValueAt (
      int rowIndex,
      int columnIndex )
  {
    double  value = parameters[ rowIndex ][ columnIndex ];
    if (Double.isNaN( value ))
      return  null;
    else
      return  String.valueOf( value );
  }

  public void setValueAt (
      Object aValue,
      int rowIndex,
      int columnIndex)
  {
    double  dvalue;
    if (aValue instanceof Number)
      dvalue = ((Number) aValue).doubleValue();
    else if (aValue instanceof String)
    {
      try
      {
```

```
        dvalue = Double.valueOf( (String) aValue ).doubleValue();
      }
      catch (NumberFormatException ex)
      {
        dvalue = Double.NaN;
      }
    }
    else
      dvalue = Double.NaN;
    if (Double.isNaN( dvalue) && rowIndex < sensorNames.length)
      if ( rowIndex == columnIndex)
        dvalue = 1;
      else
        dvalue = 0;

    parameters[ rowIndex ][ columnIndex ] = dvalue;
    fireTableCellUpdated( rowIndex, columnIndex );
  }

  public void  setParameters (
      double[][]  parameters )
  {
    this.parameters = parameters;
    this.fireTableStructureChanged();
  }

  public double  getParameter (
      int rowIndex,
      int columnIndex )
  {
    return  parameters[rowIndex][columnIndex];
  }
};

private ModelParameters  modelParameters = new ModelParameters();
private String[]  functionNames =
{ "log linear splice", "hyperbolic", "x + a log x + b" };
private LogicleFunction[]  logicleFunctions =
{ logLinearSplice,     hyperbolic,   logPlus };
private DefaultComboBoxModel functionModel = new DefaultComboBoxModel(
functionNames );
private static final double  ln_10 = Math.log( 10 );
private final String  SPECTRUM_ERROR = "The data are not compatible with the
spectrum.";
private final int  FCS_TIMER_TICK = 200;

private void  interpolate (
    double[]  func,
    int  i,
    int  n,
    double  x )
{
  int  m = n / 2;
  double  y = Math.sqrt( func[i] * x );

  func[ i + m ] = y;

  if (m > 1 )
  {
    interpolate( func, i, m, y );
    interpolate( func, i + m, m, x );
  }
```

```
}

private void  shuffle (
    double[]  sequence )
{
  for (int  i = 0;  i < sequence.length;  ++i)
  {
    int j = i + (int) Math.floor( Math.random() * (sequence.length - i) );
    double  t = sequence[i];
    sequence[i] = sequence[j];
    sequence[j] = t;
  }
}

public void  invert (
    double[][]  matrix )
{
  int  row = 0, col = 0, n = matrix.length;
  int  pivot[] = new int[n];
  int  row_index[] = new int[n];
  int  col_index[] = new int[n];

  for (int  i = 0;  i < n;  ++i)
  {
    double  big = 0;
    for (int  j = 0;  j < n;  ++j)
    {
      if (pivot[j] != 1)
        for (int  k = 0;  k < n;  ++k)
        {
          if (pivot[k] == 0)
          {
            double  abs = Math.abs(matrix[j][k]);
            if (abs >= big)
            {
              big = abs;
              row = j;
              col = k;
            }
          }
          else if (pivot[k] > 1)
            throw  new IllegalArgumentException( "matrix is singular" );
        }
    }
    ++pivot[col];
    row_index[i] = row;
    col_index[i] = col;

    if (row != col)
      for (int  k = 0;  k < n;  ++k)
      {
        double  t = matrix[row][k];
        matrix[row][k] = matrix[col][k];
        matrix[col][k] = t;
      }

    if (matrix[col][col] == 0)
      throw  new IllegalArgumentException( "matrix is singular" );
    double inverse = 1/matrix[col][col];
    matrix[col][col] = 1;
    for (int  j = 0;  j < n;  ++j)
      matrix[col][j] *= inverse;
    for (int  j = 0;  j < n;  ++j)
```

```
      if (j != col)
      {
        double  t = matrix[j][col];
        matrix[j][col] = 0;
        for (int  k = 0;  k < n;  ++k)
          matrix[j][k] -= matrix[col][k] * t;
      }
    }

    for (int  i = n-1;  i >= 0;  --i)
      if (row_index[i] != col_index[i])
        for (int  j = 0;  j < n;  ++j)
        {
          double  t = matrix[j][row_index[i]];
          matrix[j][row_index[i]] = matrix[j][col_index[i]];
          matrix[j][col_index[i]] = t;
        }
  }

  void make_busy ()
  {
    spectrumButton.setEnabled( false );
    spectrumMenuItem.setEnabled( false );

    openButton.setEnabled( false );
    openMenuItem.setEnabled( false );

    saveButton.setEnabled( false );
    saveButton.setEnabled( false );

    progressBar.setEnabled( false );
  }

  void make_ready ()
  {
    spectrumButton.setEnabled( true );
    spectrumMenuItem.setEnabled( true );

    if (sensorNames != null)
    {
      openButton.setEnabled( true );
      openMenuItem.setEnabled( true );
    }

    if (fcs != null)
    {
      saveButton.setEnabled( true );
      saveButton.setEnabled( true );
      statusBar.setText( fcs.getFile().getName() );
    }
    else
      statusBar.setText( "" );

    progressBar.setValue( 0 );
    progressBar.setEnabled( false );
  }

  void spectrum_action (
      ActionEvent e )
  {
    if (spectrumTable.isEditing())
      spectrumTable.getCellEditor().stopCellEditing();
```

```
    fileChooser.setDialogTitle( "Load Spectrum File" );
    int returnVal = fileChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      make_busy();
      try
      {
        BufferedReader  br = new BufferedReader( new FileReader(
fileChooser.getSelectedFile() ) );
        br.readLine();
        br.readLine();
        StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
        ArrayList  al = new ArrayList();
        while (st.hasMoreTokens())
          al.add( st.nextToken() );
        sensorNames = (String[]) al.toArray( stringArray );

        double[][]  parameters = new double[sensorNames.length +
2][sensorNames.length];
        int  r;
        for (r = 0;  r < sensorNames.length;  ++r)
        {
          st = new StringTokenizer( br.readLine(), "\t" );
          for (int  c = 0;  c < sensorNames.length;  ++c)
            parameters[r][c] =  Double.parseDouble( st.nextToken() );
        }

        for (;  r < parameters.length;  ++r)
          for (int  c = 0;  c < sensorNames.length;  ++c)
            parameters[r][c] =  Double.NaN;
        modelParameters.setParameters( parameters );

        if (fcs != null)
          for (r = 0;  r < sensorNames.length;  ++r)
            if (fcs.getParameter( sensorNames[r] ) == null)
            {
              fcs = null;
              break;
            }
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
      make_ready();
    }
  }

  FCSFile  fcs_data;
  FCSHandler  fcs_handler;
  IOException  fcs_exception;

  Timer  fcs_timer = new Timer( FCS_TIMER_TICK, new ActionListener ()
  {
    public void actionPerformed (
        ActionEvent  action )
    {
      progressBar.setValue( fcs_handler.getEvent() );
    }
  } );

  void data_action(ActionEvent e)
  {
```

```
if (spectrumTable.isEditing())
  spectrumTable.getCellEditor().stopCellEditing();

fileChooser.setDialogTitle( "Load FCS Data File" );
int returnVal = fileChooser.showOpenDialog( this );
if(returnVal == JFileChooser.APPROVE_OPTION)
{
  try
  {
    File  f = fileChooser.getSelectedFile();

    fcs_data = new FCSFile( f );
    for (int  i = 0;  i < sensorNames.length;  ++i)
      if (fcs_data.getParameter( sensorNames[i] ) == null)
      {
        JOptionPane.showMessageDialog( this, SPECTRUM_ERROR,
            "Error", JOptionPane.ERROR_MESSAGE );
        return;
      }

    make_busy();
    progressBar.setMinimum( 0 );
    progressBar.setValue( 0 );
    progressBar.setMaximum( fcs_data.getTotal() );
    progressBar.setEnabled( true );
    statusBar.setText( fcs_data.getFile().getName() );

    new Thread( data_thread, "FCS reader" ).start();
    fcs_timer.start();
  }
  catch (Exception ex)
  {
    ex.printStackTrace();
    System.exit( 1 );
  }
 }
}

Runnable  data_thread = new Runnable()
{
  public void run ()
  {
    Thread.currentThread().setPriority( Thread.MIN_PRIORITY );
    fcs_exception = null;
    int  n = fcs_data.getTotal();
    int  m = fcs_data.getParameters();
    int[][]  int_value = new int[m][n];
    try
    {
      fcs_handler = fcs_data.getInputIterator();
      for (int  i = 0;  fcs_handler.hasMoreEvents();  ++i)
        for (int  j = 0;  fcs_handler.hasMoreValues();  ++j)
          int_value[j][i] = fcs_handler.readValue();
      fcs_handler.close();
      originalData = int_value;
    }
    catch (IOException ex)
    {
      ex.printStackTrace();
      fcs_exception = ex;
    }
    SwingUtilities.invokeLater( data_complete );
  }
```

```
};

Runnable  data_complete = new Runnable ()
{
  public void run ()
  {
    fcs_timer.stop();
    if (fcs_exception == null)
    {
      fcs = fcs_data;
    }
    make_ready();
  }
};

void save_action(ActionEvent e)
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();

  fileChooser.setDialogTitle( "Save New FCS Data File" );
  int returnVal = fileChooser.showSaveDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      File  f = fileChooser.getSelectedFile();
      logicle = logicleFunctions[ functionChoice.getSelectedIndex() ];

      if (f.exists())
      {
        Object[]  message = { f, "already exists. Do you want to replace it?" };
        if (JOptionPane.OK_OPTION !=
            JOptionPane.showConfirmDialog( this, message, "Overwrite",
            JOptionPane.OK_CANCEL_OPTION, JOptionPane.WARNING_MESSAGE ))
          return;
      }

      int  n = fcs.getTotal();
      int  m = sensorNames.length;
      rawData = null;
      compensatedData = null;
      newData = null;

      compensationMatrix = new double[m][m];
      for (int  i = 0;  i < m; ++i)
        for (int  j = 0;  j < m;  ++j)
          compensationMatrix[j][i] = modelParameters.getParameter( i, j );
      try
      {
        invert( compensationMatrix );
      }
      catch (IllegalArgumentException ex)
      {
        JOptionPane.showMessageDialog( this, "Spectrum Matrix is Singular",
"Error",
            JOptionPane.ERROR_MESSAGE );
        return;
      }

      make_busy();

      progress = 0;
```

```
        progressBar.setMinimum( 0 );
        progressBar.setMaximum( 2 * m + m * m  );
        progressBar.setValue( 0 );
        progressBar.setEnabled( true );
        statusBar.setText( "Working" );

        fcs_data = new FCSFile(  fileChooser.getSelectedFile() );
        fcs_data.getTextSegment().readFrom( fcs.getTextSegment() );

        new Thread( xfrm_thread, "Transform" ).start();
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }

  Runnable  xfrm_thread = new Runnable()
  {
    public void run ()
    {
      try
      {
        int  n = fcs.getTotal();
        int  m = sensorNames.length;

        rawData = new double[m][n];
        for (int  i = 0;  i < m;  ++i)
        {
          FCSParameter  p = fcs.getParameter( sensorNames[i] );
          int[]  input_data = originalData[p.getIndex()-1];
          double[]  output_data = rawData[i];
          if (p.isLog())
          {
            double[]  antilog = new double[ p.getRange() ];
            antilog[0] = p.getMinimum();
            interpolate( antilog, 0, p.getRange(), p.getMaximum() );
            double[]  fuzz = new double[ 1<<FUZZY_BITS ];
            fuzz[0] = 1;
            interpolate( fuzz, 0, 1<<FUZZY_BITS, antilog[1]/antilog[0] );
            shuffle( fuzz );
            for (int  j = 0;  j < n;  ++j)
              output_data[j] = antilog[ input_data[j] ] * fuzz[ j &
(1<<FUZZY_BITS-1) ];
          }
          else
          {
            double  g = p.getGain();
            for (int  j = 0;  j < n;  ++j)
              output_data[j] = input_data[j] / g;
          }
          ++progress;
          SwingUtilities.invokeLater( progress_made );
        }

        compensatedData = new double[m][n];
        for (int  i = 0;  i < m;  ++i)
        {
          double[]  output_data = compensatedData[i];
          for (int  j = 0;  j < m;  ++j)
          {
            double[]  input_data = rawData[j];
```

```
          double  coefficient = compensationMatrix[i][j];
          for (int  k = 0;  k < n;  ++k)
            output_data[k] += input_data[k] * coefficient;
          ++progress;
          SwingUtilities.invokeLater( progress_made );
        }
      }

      newData = new int[m][n];
      for (int  i = 0;  i < m;  ++i)
      {
        FCSParameter  oldp = fcs.getParameter( sensorNames[i] );
        FCSParameter  newp = fcs_data.addParameter();
        newp.setAttribute( "$P", "N", "["+sensorNames[i]+"]" );
        newp.setAttribute( "$P", "S", oldp.getAttribute( "$P", "S" ) );
        newp.setBits( oldp.getBits() );
        newp.setRange( oldp.getRange() );
        newp.setDecades( oldp.getDecades(), oldp.getMinimum() );
        newp.setGain( oldp.getGain() );

        double[]  input_data = compensatedData[i];
        int[]  transformed_data = newData[i];
        if (newp.isLog())
        {
          double[]  lookup = logicle.transform( i, newp );
          for (int  j = 0;  j < n;  ++j)
          {
            int  k = Arrays.binarySearch( lookup, input_data[j] );
            if (k < 0)
              transformed_data[j] = -k-1;
            else
              transformed_data[j] = k;
          }
        }
        else
        {
          double  g = newp.getGain();
          for (int  j = 0;  j < n;  ++i)
            transformed_data[j] = (int) Math.round( input_data[j] * g );
        }
        ++progress;
        SwingUtilities.invokeLater( progress_made );
      }

      compensatedData = null;
      SwingUtilities.invokeLater( xfrm_complete );
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
};

Runnable  progress_made = new Runnable ()
{
  public void run ()
  {
    progressBar.setValue( progress );
  }
};

Runnable  xfrm_complete = new Runnable ()
```

```
{
  public void  run ()
  {
    progressBar.setMinimum( 0 );
    progressBar.setValue( 0 );
    progressBar.setMaximum( fcs_data.getTotal() );
    progressBar.setEnabled( true );
    statusBar.setText( fcs_data.getFile().getName() );

    new Thread( save_thread, "FCS writer" ).start();
    fcs_timer.start();
  }
};

Runnable  save_thread = new Runnable()
{
  public void run ()
  {
    Thread.currentThread().setPriority( Thread.MIN_PRIORITY );
    fcs_exception = null;
    try
    {
      fcs_handler = fcs_data.getOutputIterator();
      for (int  i = 0;  fcs_handler.hasMoreEvents();  ++i)
      {
        for (int  j = 0;  j < originalData.length;  ++j)
          fcs_handler.writeValue( originalData[j][i] );
        for (int  j = 0;  j < newData.length;  ++j)
          fcs_handler.writeValue( newData[j][i] );
      }
      fcs_handler.close();
    }
    catch (IOException ex)
    {
      ex.printStackTrace();
      fcs_exception = ex;
    }
    SwingUtilities.invokeLater( save_complete );
  }
};

Runnable  save_complete = new Runnable ()
{
  public void run ()
  {
    fcs_timer.stop();
    if (fcs_exception == null)
    {
    }
    make_ready();
  }
};
private JLabel jLabel8 = new JLabel();
private Component component1;
private Component component4;

void function_action(ActionEvent e)
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();

  for (int  i = 0;  i < sensorNames.length;  ++i)
    for (int  j = 0;  j < this.N_PARAMETERS;  ++j)
```

```
        modelParameters.setValueAt( null, sensorNames.length + j, i );
    }

    void help_action(ActionEvent e)
    {
      helpDialog.setLocationRelativeTo( this );
      helpDialog.setVisible( true );
    }

    void help_dismiss(ActionEvent e)
    {
      helpDialog.setVisible( false );
    }

    void about_action(ActionEvent e)
    {
      JOptionPane.showMessageDialog( this, aboutMessage, "About Logicle",
JOptionPane.PLAIN_MESSAGE );
    }
}
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.Random;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;
import javax.swing.event.*;
import com.borland.jbcl.layout.*;
import javax.swing.border.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame
extends JFrame implements TableModelListener
{
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton openButton = new JButton();
  private JButton saveButton = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrumTable = new JTable();
  private JMenuItem spectrumMenuItem = new JMenuItem();
  private JFileChooser  fileChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );
  private JMenuItem openMenuItem = new JMenuItem();
  private JMenuItem saveMenuItem = new JMenuItem();
  private JButton spectrumButton = new JButton();
  private JComboBox functionChoice = new JComboBox();
  private JLabel jLabel1 = new JLabel();
  private JDialog helpDialog = new JDialog();
  private JPanel jPanel1 = new JPanel();
  private JButton helpDismiss = new JButton();
  private JScrollPane jScrollPane2 = new JScrollPane();
  private JEditorPane helpPane = new JEditorPane();
```

```
private JPanel aboutMessage = new JPanel();
private JLabel jLabel2 = new JLabel();
private JLabel jLabel3 = new JLabel();
private GridLayout gridLayout1 = new GridLayout();
private JLabel jLabel4 = new JLabel();
private Component component2;
private JLabel jLabel5 = new JLabel();
private JLabel jLabel6 = new JLabel();
private Border border1;
private Component component3;
private JLabel jLabel7 = new JLabel();
private JPanel jPanel2 = new JPanel();
private JLabel statusBar = new JLabel();
private JProgressBar progressBar = new JProgressBar();
private BorderLayout borderLayout3 = new BorderLayout();
private BorderLayout borderLayout2 = new BorderLayout();
private JMenuItem jMenuItem2 = new JMenuItem();
private JLabel jLabel8 = new JLabel();
private Component component1;
private Component component4;
private JPanel errorMessage = new JPanel();
private GridLayout gridLayout2 = new GridLayout();
private JLabel errorLabel1 = new JLabel();
private JLabel errorLabel2 = new JLabel();
private AxisDialog  axisDialog;

//Construct the frame
public ConverterFrame()
{
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try
  {
    jbInit();
    spectrumTable.setModel( modelParameters );
    modelParameters.addTableModelListener( new TableModelListener ()
        {
      public void tableChanged(TableModelEvent e)
      {
        if (spectrumTable.getSelectedColumn() >= 0 && e.getColumn() ==
spectrumTable.getSelectedColumn())
          nomogram_action();
      }
        } );
    spectrumTable.getTableHeader().setReorderingAllowed( false );
    spectrumTable.setColumnSelectionAllowed( true );
    spectrumTable.setRowSelectionAllowed( false );
    spectrumTable.getColumnModel().getSelectionModel().setSelectionMode(
ListSelectionModel.SINGLE_SELECTION );
    spectrumTable.getColumnModel().addColumnModelListener(
    new TableColumnModelListener ()
    {
      public void columnSelectionChanged( ListSelectionEvent  lse )
      {
        nomogram_action();
      }
      public void columnMarginChanged( ChangeEvent  ce )
      {
      }
      public void columnMoved( TableColumnModelEvent  tcme )
      {
      }
      public void columnRemoved( TableColumnModelEvent  tcme )
      {
```

```
        }
        public void columnAdded( TableColumnModelEvent  tcme )
        {
        }
      } );
      axisDialog = new AxisDialog( this, "Logicle Scale", false );
      axisDialog.pack();
      helpPane.setPage( ConverterFrame.class.getResource( "help.html" ) );
      helpDialog.pack();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
  }
  //Component initialization
  private void jbInit() throws Exception  {
    image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
    image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
    image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    component2 = Box.createVerticalStrut(8);
    border1 = BorderFactory.createEmptyBorder(10,15,10,25);
    component3 = Box.createVerticalStrut(8);
    component1 = Box.createVerticalStrut(8);
    component4 = Box.createVerticalStrut(8);
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Logicle Fluorescence Compensation");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About ...");
    jMenuHelpAbout.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        about_action(e);
      }
    });
    openButton.setIcon(image1);
    openButton.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    openButton.setEnabled(false);
    openButton.setToolTipText("Open FCS File");
    saveButton.setIcon(image2);
    saveButton.addActionListener(new java.awt.event.ActionListener()
```

```
    {
      public void actionPerformed(ActionEvent e)
      {
        save_action(e);
      }
    });
    saveButton.setEnabled(false);
    saveButton.setToolTipText("Save Extended FCS File");
    jButton3.setIcon(image3);
    jButton3.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        help_action(e);
      }
    });
    jButton3.setToolTipText("Help");
    spectrumMenuItem.setIcon(new
ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
    spectrumMenuItem.setText("Spectrum ...");
    spectrumMenuItem.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        spectrum_action(e);
      }
    });
    openMenuItem.setEnabled(false);
    openMenuItem.setIcon(image1);
    openMenuItem.setText("Data ...");
    openMenuItem.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    saveMenuItem.setEnabled(false);
    saveMenuItem.setIcon(image2);
    saveMenuItem.setText("Save As ...");
    saveMenuItem.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        save_action(e);
      }
    });
    functionChoice.setFont(new java.awt.Font("Dialog", 1, 14));
    functionChoice.setMaximumSize(new Dimension(32767, 27));
    functionChoice.setMinimumSize(new Dimension(138, 27));
    functionChoice.setPreferredSize(new Dimension(142, 27));
    functionChoice.setToolTipText("Select a Data Transformation");
    functionChoice.setModel(functionModel);
    functionChoice.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        function_action(e);
      }
    });
    spectrumButton.setMaximumSize(new Dimension(51, 27));
    spectrumButton.setMinimumSize(new Dimension(51, 27));
    spectrumButton.setToolTipText("Load Spectrum Matrix");
    spectrumButton.setIcon(new
ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
```

```
    spectrumButton.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        spectrum_action(e);
      }
    });
    jLabel1.setPreferredSize(new Dimension(17, 17));
    spectrumTable.setRowSelectionAllowed(false);
    helpDismiss.setText("Close");
    helpDismiss.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        help_dismiss(e);
      }
    });
    helpDialog.setTitle("Logicle Help");
    helpDialog.getContentPane().setLayout(borderLayout2);
    jMenuItem2.setText("Help ...");
    jMenuItem2.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        help_action(e);
      }
    });
    helpPane.setPreferredSize(new Dimension(400, 400));
    helpPane.setEditable(false);
    helpPane.setText("<html>\r\n  <head>\r\n\r\n  </head>\r\n  <body>\r\n    <p>\r\n
Logicle Data Converter " +
    "    \r\n    </p>\r\n  </body>\r\n</html>\r\n");
    helpPane.setContentType("text/html");
    jScrollPane2.setPreferredSize(new Dimension(400, 400));
    jLabel2.setFont(new java.awt.Font("Dialog", 1, 14));
    jLabel2.setAlignmentY((float) 0.0);
    jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel2.setHorizontalTextPosition(SwingConstants.LEADING);
    jLabel2.setIconTextGap(0);
    jLabel2.setText("Logicle Fluorescence Compensation");
    jLabel3.setAlignmentX((float) 0.5);
    jLabel3.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel3.setIconTextGap(0);
    jLabel3.setText("Wayne A. Moore");
    aboutMessage.setLayout(gridLayout1);
    gridLayout1.setColumns(1);
    gridLayout1.setRows(11);
    jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel4.setIconTextGap(0);
    jLabel4.setText("October 2002");
    jLabel5.setFont(new java.awt.Font("Dialog", 2, 10));
    jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel5.setHorizontalTextPosition(SwingConstants.LEADING);
    jLabel5.setIconTextGap(0);
    jLabel5.setText("Copyright 2002, by The Board of Trustees");
    jLabel6.setFont(new java.awt.Font("Dialog", 2, 10));
    jLabel6.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel6.setHorizontalTextPosition(SwingConstants.LEADING);
    jLabel6.setIconTextGap(0);
    jLabel6.setText("of the Leland Stanford Jr. University");
    aboutMessage.setBorder(border1);
    aboutMessage.setMinimumSize(new Dimension(60, 40));
    aboutMessage.setPreferredSize(new Dimension(300, 250));
```

```
jLabel7.setHorizontalAlignment(SwingConstants.CENTER);
jLabel7.setText("David R. Parks");
statusBar.setText(" ");
jPanel2.setLayout(borderLayout3);
progressBar.setString("Compensate");
jLabel8.setHorizontalAlignment(SwingConstants.CENTER);
jLabel8.setText("Version 0.6");
errorMessage.setLayout(gridLayout2);
gridLayout2.setColumns(1);
gridLayout2.setRows(0);
errorLabel1.setHorizontalAlignment(SwingConstants.CENTER);
errorLabel2.setHorizontalAlignment(SwingConstants.CENTER);
errorLabel2.setText("jLabel10");
jMenu1.setText("View");
jMenuItem1.setText("Scale");
jMenuItem1.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    scale_action(e);
  }
});
jToolBar.add(spectrumButton, null);
jToolBar.add(openButton);
jToolBar.add(saveButton);
jToolBar.add(jButton3);
jToolBar.add(jLabel1, null);
jToolBar.add(functionChoice, null);
jMenuFile.add(spectrumMenuItem);
jMenuFile.add(openMenuItem);
jMenuFile.add(saveMenuItem);
jMenuFile.addSeparator();
jMenuFile.add(jMenuFileExit);
jMenuHelp.add(jMenuItem2);
jMenuHelp.addSeparator();
jMenuHelp.add(jMenuHelpAbout);
jMenuBar1.add(jMenuFile);
jMenuBar1.add(jMenu1);
jMenuBar1.add(jMenuHelp);
this.setJMenuBar(jMenuBar1);
contentPane.add(jToolBar,  BorderLayout.NORTH);
contentPane.add(jScrollPane1, BorderLayout.CENTER);
contentPane.add(jPanel2, BorderLayout.SOUTH);
jPanel2.add(statusBar,  BorderLayout.CENTER);
jPanel2.add(progressBar,  BorderLayout.EAST);
jScrollPane1.getViewport().add(spectrumTable, null);
helpDialog.getContentPane().add(jPanel1, BorderLayout.SOUTH);
jPanel1.add(helpDismiss, null);
helpDialog.getContentPane().add(jScrollPane2, BorderLayout.CENTER);
jScrollPane2.getViewport().add(helpPane, null);
aboutMessage.add(jLabel2, null);
aboutMessage.add(component4, null);
aboutMessage.add(jLabel8, null);
aboutMessage.add(component1, null);
aboutMessage.add(jLabel3, null);
aboutMessage.add(jLabel7, null);
aboutMessage.add(jLabel4, null);
aboutMessage.add(component2, null);
aboutMessage.add(jLabel5, null);
aboutMessage.add(jLabel6, null);
aboutMessage.add(component3, null);
errorMessage.add(errorLabel1, null);
errorMessage.add(errorLabel2, null);
```

```
  jMenu1.add(jMenuItem1);
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
}
//Help | About action performed
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
  }
}

private static final String[]  stringArray = new String[0];
private static final int  N_PARAMETERS = 2;
private static final int FUZZY_BITS = 8;
private FCSFile  fcs;
private String[]  sensorNames = new String[0];
private int[][]  originalData;
private int[][]  newData;
private double[][]  rawData;
private double[][]  compensatedData;
private double[][]  compensationMatrix;
private int  progress;
protected LogicleFunction  logicle;
protected LogicleFunction  logarithm = new LogicleFunction();

public class  LogicleFunction
{
  double  decades, minimum, maximum;

  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double[]  lookup = new double[ parameter.getRange() ];
    maximum = parameter.getMaximum();
    minimum = parameter.getMinimum();
    decades = parameter.getDecades();
    lookup[0] = minimum;
    interpolate( lookup, 0, lookup.length, maximum );

    return  lookup;
  }
}

public final  LogicleFunction  logLinearSplice = new LogicleFunction ()
{

  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double  linear_range = modelParameters.getParameter( sensorNames.length,
column );
    if (Double.isNaN( linear_range ))
      return  super.transform( column, parameter );

    maximum = parameter.getMaximum();
    double ln_range = Math.log( maximum / linear_range ) + 2;
    decades = ln_range / ln_10;
```

```
      minimum = maximum / Math.exp( ln_range );

      double[]  lookup = new double[ parameter.getRange() ];
      lookup[0] = minimum;
      interpolate( lookup, 0, lookup.length, maximum );

      int  splice = (int) Math.round(
          Math.log( linear_range / minimum ) * (double) parameter.getRange() /
parameter.getDecades() / ln_10 );
      if (splice > 0 && splice < parameter.getRange() - 1)
      {
        double  delta = (lookup[splice+1] - lookup[splice-1])/2;
        double  base = lookup[splice];
        for (int  j = 0;  j < splice;  ++j)
          lookup[j] = base - (splice - j) * delta;
      }

      return  lookup;
    }
  };

  public final  LogicleFunction  biexponential = new LogicleFunction ()
  {

    public double[]  transform (
        int  column,
        FCSParameter  parameter )
    {
      double  width = modelParameters.getParameter( sensorNames.length, column );
      if (Double.isNaN( width ))
        return  super.transform( column, parameter );

      decades = parameter.getDecades();
      width /= decades;

      double  extra = modelParameters.getParameter( sensorNames.length + 1, column
);
      if (Double.isNaN( extra ) || extra < 0)
        extra = 0;
      int  zero = (int) Math.round( extra * parameter.getRange() / (extra + decades)
);
      if (zero > 0)
      {
        extra = zero * (decades + extra) / parameter.getRange(); // so extra falls
exactly on channel boundry
        decades = extra * parameter.getRange() / zero;
      }

      maximum = parameter.getMaximum();
      double positive_range = ln_10 * decades;
      minimum = maximum / Math.exp( positive_range );

      double[]  positive = new double[ parameter.getRange() ];
      positive[0] = 1;
      interpolate( positive, 0, positive.length, Math.exp( positive_range ) );

      double negative_range = root( positive_range, width );
      System.out.println( positive_range );
      System.out.println( negative_range );
      double[]  negative = new double[ positive.length ];
      negative[0] = 1;
      interpolate( negative, 0, negative.length, Math.exp( -negative_range ) );
```

```
      double  s = Math.exp( (positive_range + negative_range) * (width - extra /
decades) );
      for (int  j = 0;  j < negative.length;  ++j)
        negative[j] *= s;

      s = positive[zero] - negative[zero];
      for (int  j = zero;  j < positive.length;  ++j)
        positive[j] = minimum * (positive[j] - negative[j] - s);

      for (int  j = 0;  j < zero;  ++j)
        positive[j] = -positive[2 * zero - j];

      return  positive;
    }
  };


  public final  LogicleFunction  hyperbolic = new LogicleFunction ()
  {

    public double[]  transform (
        int  column,
        FCSParameter  parameter )
    {
      double  linear_range = modelParameters.getParameter( sensorNames.length,
column );
      if (Double.isNaN( linear_range ))
        return  super.transform( column, parameter );

      maximum = parameter.getMaximum();
      double  scale_factor = linear_range / (Math.E - 1/Math.E);
      double ln_range = Math.log( maximum / scale_factor ) + 2;
      decades = ln_range / ln_10;
      minimum = maximum / Math.exp( ln_range );

      double[]  lookup = new double[ parameter.getRange() ];
      lookup[0] = 1 / Math.E;
      interpolate( lookup, 0, lookup.length, maximum / scale_factor );

      for (int  j = 0;  j < lookup.length;  ++j)
        lookup[j] = (lookup[j] - 1/lookup[j]) * scale_factor;

      return  lookup;
    }
  };

  public final  LogicleFunction  logPlus = new LogicleFunction ()
  {
    public double[]  transform (
        int  column,
        FCSParameter  parameter )
    {
      double  parameter1 = modelParameters.getParameter( sensorNames.length, column
);
      double  parameter2 = modelParameters.getParameter( sensorNames.length + 1,
column );
      if (Double.isNaN( parameter1 ))
        return  super.transform( column, parameter );
      if (Double.isNaN( parameter2 ))
        parameter2 = 0;

      double[]  lookup = super.transform( column, parameter );
```

```
//      parameter2 += parameter1 * Math.log( parameter.getMinimum() ) / ln_10;
        parameter1 *= parameter.getDecades() / parameter.getRange();
        for (int  j = 0;  j < lookup.length;  ++j)
          lookup[j] += parameter1 * j + parameter2;

        return  lookup;
      }
    };

    private class  ModelParameters
    extends AbstractTableModel
    {
      private double[][]  parameters = new double[0][];

      public int getRowCount()
      {
        return  parameters.length;
      }

      public int getColumnCount()
      {
        return  sensorNames.length;
      }

      public String getColumnName (
          int columnIndex )
      {
        return  sensorNames[ columnIndex ];
      }

      public Class getColumnClass (
          int columnIndex )
      {
        return  super.getColumnClass( columnIndex );
      }

      public boolean isCellEditable (
          int rowIndex,
          int columnIndex )
      {
        return  true;
      }

      public Object getValueAt (
          int rowIndex,
          int columnIndex )
      {
        double  value = parameters[ rowIndex ][ columnIndex ];
        if (Double.isNaN( value ))
          return  null;
        else
          return  String.valueOf( value );
      }

      public void setValueAt (
          Object aValue,
          int rowIndex,
          int columnIndex)
      {
        double  dvalue;
        if (aValue instanceof Number)
          dvalue = ((Number) aValue).doubleValue();
        else if (aValue instanceof String)
```

```
    {
      try
      {
        dvalue = Double.valueOf( (String) aValue ).doubleValue();
      }
      catch (NumberFormatException ex)
      {
        dvalue = Double.NaN;
      }
    }
    else
      dvalue = Double.NaN;
    if (Double.isNaN( dvalue) && rowIndex < sensorNames.length)
      if ( rowIndex == columnIndex)
        dvalue = 1;
      else
        dvalue = 0;

    parameters[ rowIndex ][ columnIndex ] = dvalue;
    fireTableCellUpdated( rowIndex, columnIndex );
  }

  public void  setParameters (
      double[][]  parameters )
  {
    this.parameters = parameters;
    this.fireTableStructureChanged();
  }

  public double  getParameter (
      int rowIndex,
      int columnIndex )
  {
    return  parameters[rowIndex][columnIndex];
  }
};

private ModelParameters  modelParameters = new ModelParameters();
private String[]  functionNames =
{ "biexponential", "log linear splice", "hyperbolic", "x + a log x + b" };
private LogicleFunction[]  logicleFunctions =
{ biexponential, logLinearSplice,     hyperbolic,   logPlus };
private DefaultComboBoxModel functionModel = new DefaultComboBoxModel(
functionNames );
private static final double  ln_10 = Math.log( 10 );
private final String  SPECTRUM_ERROR = "The data are not compatible with the
spectrum.";
private final int  FCS_TIMER_TICK = 200;
private final Random  random = new Random();

private void  interpolate (
    double[]  func,
    int  i,
    int  n,
    double  x )
{
  int  m = n / 2;
  double  y = Math.sqrt( func[i] * x );

  func[ i + m ] = y;

  if (m > 1 )
  {
```

```
    interpolate( func, i, m, y );
    interpolate( func, i + m, m, x );
  }

}

private void  shuffle (
    double[]  sequence )
{
  for (int  i = 0;  i < sequence.length;  ++i)
  {
    int j = i + random.nextInt( sequence.length - i );
    double  t = sequence[i];
    sequence[i] = sequence[j];
    sequence[j] = t;
  }
}

public void  invert (
    double[][]  matrix )
{
  int  row = 0, col = 0, n = matrix.length;
  int  pivot[] = new int[n];
  int  row_index[] = new int[n];
  int  col_index[] = new int[n];

  for (int  i = 0;  i < n;  ++i)
  {
    double  big = 0;
    for (int  j = 0;  j < n;  ++j)
    {
      if (pivot[j] != 1)
        for (int  k = 0;  k < n;  ++k)
        {
          if (pivot[k] == 0)
          {
            double  abs = Math.abs(matrix[j][k]);
            if (abs >= big)
            {
              big = abs;
              row = j;
              col = k;
            }
          }
          else if (pivot[k] > 1)
            throw  new IllegalArgumentException( "matrix is singular" );
        }
    }
    ++pivot[col];
    row_index[i] = row;
    col_index[i] = col;

    if (row != col)
      for (int  k = 0;  k < n;  ++k)
      {
        double  t = matrix[row][k];
        matrix[row][k] = matrix[col][k];
        matrix[col][k] = t;
      }

    if (matrix[col][col] == 0)
      throw  new IllegalArgumentException( "matrix is singular" );
    double inverse = 1/matrix[col][col];
```

```
    matrix[col][col] = 1;
    for (int  j = 0;  j < n;  ++j)
      matrix[col][j] *= inverse;
    for (int  j = 0;  j < n;  ++j)
      if (j != col)
      {
        double  t = matrix[j][col];
        matrix[j][col] = 0;
        for (int  k = 0;  k < n;  ++k)
          matrix[j][k] -= matrix[col][k] * t;
      }
  }

  for (int  i = n-1;  i >= 0;  --i)
    if (row_index[i] != col_index[i])
      for (int  j = 0;  j < n;  ++j)
      {
        double  t = matrix[j][row_index[i]];
        matrix[j][row_index[i]] = matrix[j][col_index[i]];
        matrix[j][col_index[i]] = t;
      }
}

double  root (
    double  b,
    double  w )
{
  double  Xlo = 0;
  double  Xhi = b;

  double  Flo = Double.NEGATIVE_INFINITY;
  double  Fhi = w * 2 * b;

  double d = (Xlo + Xhi)/2;
  double Dx = Math.abs( Xlo - Xhi );
  double Dx_last = Dx;

  double Fb = -2 * Math.log(b) + w*b;
  double F = 2 * Math.log(d) + w*d + Fb;
  double DF = 2/d + w;

  if (w == 0)
    return  b;

  for (int  i = 0;  i < 100;  ++i)
  {
    if (((d-Xhi)*DF-F)*((d-Xlo)*DF-F) >= 0
        || Math.abs( 2 * F ) > Math.abs( Dx_last * DF ))
    {
      System.out.println( "bisection " + Dx );
      Dx = (Xhi - Xlo)/2;
      d = Xlo + Dx;
      if (d == Xlo)
        return  d;
    }
    else
    {
      Dx = F/DF;
      System.out.println( "Newton " + Dx );
      double  t = d;
      d -= Dx;
      if (d == t)
        return  d;
```

```
    }
    if (Math.abs( Dx ) < 1E-12)
      return  d;
    Dx_last = Dx;

    F = 2 * Math.log(d) + w*d + Fb;
    DF = 2/d + w;
    if (F < 0)
    {
      Xlo = d;
      Flo = F;
    }
    else
    {
      Xhi = d;
      Fhi = F;
    }
  }

  throw  new IllegalStateException( "exceeded maximum iterations" );
}

void make_busy ()
{
  spectrumButton.setEnabled( false );
  spectrumMenuItem.setEnabled( false );

  openButton.setEnabled( false );
  openMenuItem.setEnabled( false );

  saveButton.setEnabled( false );
  saveButton.setEnabled( false );

  progressBar.setEnabled( false );
}

void make_ready ()
{
  spectrumButton.setEnabled( true );
  spectrumMenuItem.setEnabled( true );

  if (sensorNames != null)
  {
    openButton.setEnabled( true );
    openMenuItem.setEnabled( true );
  }

  if (fcs != null)
  {
    saveButton.setEnabled( true );
    saveButton.setEnabled( true );
    statusBar.setText( fcs.getFile().getName() );
  }
  else
    statusBar.setText( "" );

  progressBar.setValue( 0 );
  progressBar.setEnabled( false );
}

void spectrum_action (
    ActionEvent e )
{
```

```
    if (spectrumTable.isEditing())
      spectrumTable.getCellEditor().stopCellEditing();

    fileChooser.setDialogTitle( "Load Spectrum File" );
    int returnVal = fileChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      make_busy();
      try
      {
        BufferedReader  br = new BufferedReader( new FileReader(
fileChooser.getSelectedFile() ) );
        br.readLine();
        br.readLine();
        StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
        ArrayList  al = new ArrayList();
        while (st.hasMoreTokens())
          al.add( st.nextToken() );
        sensorNames = (String[]) al.toArray( stringArray );

        double[][]  parameters = new double[sensorNames.length +
2][sensorNames.length];
        int  r;
        for (r = 0;  r < sensorNames.length;  ++r)
        {
          st = new StringTokenizer( br.readLine(), "\t" );
          for (int  c = 0;  c < sensorNames.length;  ++c)
            parameters[r][c] =  Double.parseDouble( st.nextToken() );
        }

        for (;  r < parameters.length;  ++r)
          for (int  c = 0;  c < sensorNames.length;  ++c)
            parameters[r][c] =  Double.NaN;
        modelParameters.setParameters( parameters );

        if (fcs != null)
          for (r = 0;  r < sensorNames.length;  ++r)
            if (fcs.getParameter( sensorNames[r] ) == null)
            {
              fcs = null;
              break;
            }
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
      make_ready();
    }
  }

  FCSFile  fcs_data;
  FCSHandler  fcs_handler;
  IOException  fcs_exception;

  Timer  fcs_timer = new Timer( FCS_TIMER_TICK, new ActionListener ()
  {
    public void actionPerformed (
        ActionEvent  action )
    {
      progressBar.setValue( fcs_handler.getEvent() );
    }
  } );
```

```
void data_action(ActionEvent e)
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();

  fileChooser.setDialogTitle( "Load FCS Data File" );
  int returnVal = fileChooser.showOpenDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      File  f = fileChooser.getSelectedFile();

      fcs_data = new FCSFile( f );
      for (int  i = 0;  i < sensorNames.length;  ++i)
        if (fcs_data.getParameter( sensorNames[i] ) == null)
        {
          JOptionPane.showMessageDialog( this, SPECTRUM_ERROR,
              "Error", JOptionPane.ERROR_MESSAGE );
          return;
        }

       make_busy();
       progressBar.setMinimum( 0 );
       progressBar.setValue( 0 );
       progressBar.setMaximum( fcs_data.getTotal() );
       progressBar.setEnabled( true );
       statusBar.setText( fcs_data.getFile().getName() );

       new Thread( data_thread, "FCS reader" ).start();
       fcs_timer.start();
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
        System.exit( 1 );
      }
    }
  }

  Runnable  data_thread = new Runnable()
  {
    public void run ()
    {
      Thread.currentThread().setPriority( Thread.MIN_PRIORITY );
      fcs_exception = null;
      int  n = fcs_data.getTotal();
      int  m = fcs_data.getParameters();
      int[][]  int_value = new int[m][n];
      try
      {
        fcs_handler = fcs_data.getInputIterator();
        for (int  i = 0;  fcs_handler.hasMoreEvents();  ++i)
          for (int  j = 0;  fcs_handler.hasMoreValues();  ++j)
            int_value[j][i] = fcs_handler.readValue();
        fcs_handler.close();
        originalData = int_value;
       }
       catch (IOException ex)
       {
         fcs_exception = ex;
       }
```

```
    SwingUtilities.invokeLater( data_complete );
  }
};

Runnable  data_complete = new Runnable ()
{
  public void run ()
  {
    fcs_timer.stop();
    if (fcs_exception != null)
    {
      errorLabel1.setText( fcs_exception.getMessage() );
      errorLabel2.setText( fcs_data.getFile().getPath() );
      JOptionPane.showMessageDialog( ConverterFrame.this, errorMessage,
                                     "FCS Error", JOptionPane.ERROR_MESSAGE );

    }
    else
    {
      fcs = fcs_data;
    }
    make_ready();
  }
};

void save_action(ActionEvent e)
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();

  fileChooser.setDialogTitle( "Save New FCS Data File" );
  int returnVal = fileChooser.showSaveDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      File  f = fileChooser.getSelectedFile();
      logicle = logicleFunctions[ functionChoice.getSelectedIndex() ];

      if (f.exists())
      {
        Object[]  message = { f, "already exists. Do you want to replace it?" };
        if (JOptionPane.OK_OPTION !=
            JOptionPane.showConfirmDialog( this, message, "Overwrite",
            JOptionPane.OK_CANCEL_OPTION, JOptionPane.WARNING_MESSAGE ))
          return;
      }

      int  n = fcs.getTotal();
      int  m = sensorNames.length;
      rawData = null;
      compensatedData = null;
      newData = null;

      compensationMatrix = new double[m][m];
      for (int  i = 0;  i < m; ++i)
        for (int  j = 0;  j < m;  ++j)
          compensationMatrix[j][i] = modelParameters.getParameter( i, j );
      try
      {
        invert( compensationMatrix );
      }
      catch (IllegalArgumentException ex)
      {
```

```
          JOptionPane.showMessageDialog( this, "Spectrum Matrix is Singular",
"Spectrum Error",
              JOptionPane.ERROR_MESSAGE );
          return;
        }

        make_busy();

        progress = 0;
        progressBar.setMinimum( 0 );
        progressBar.setMaximum( 2 * m + m * m  );
        progressBar.setValue( 0 );
        progressBar.setEnabled( true );
        statusBar.setText( "Working" );

        fcs_data = new FCSFile(  fileChooser.getSelectedFile() );
        fcs_data.getTextSegment().readFrom( fcs.getTextSegment() );

        new Thread( xfrm_thread, "Transform" ).start();
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }

  Runnable  xfrm_thread = new Runnable()
  {
    public void run ()
    {
      try
      {
        int  n = fcs.getTotal();
        int  m = sensorNames.length;

        rawData = new double[m][n];
        for (int  i = 0;  i < m;  ++i)
        {
          FCSParameter  p = fcs.getParameter( sensorNames[i] );
          int[]  input_data = originalData[p.getIndex()-1];
          double[]  output_data = rawData[i];
          if (p.isLog())
          {
            double[]  antilog = new double[ p.getRange() ];
            antilog[0] = p.getMinimum();
            interpolate( antilog, 0, p.getRange(), p.getMaximum() );

            double[]  fuzz = new double[ 1<<FUZZY_BITS ];
            fuzz[0] = 1;
            interpolate( fuzz, 0, 1<<FUZZY_BITS, antilog[1]/antilog[0] );
            shuffle( fuzz );

            for (int  j = 0;  j < n;  ++j)
              output_data[j] = antilog[ input_data[j] ] * fuzz[ j &
((1<<FUZZY_BITS)-1) ];
          }
          else
          {
            double  g = p.getGain();
            for (int  j = 0;  j < n;  ++j)
              output_data[j] = input_data[j] / g;
          }
```

```
  ++progress;
  SwingUtilities.invokeLater( progress_made );
}

compensatedData = new double[m][n];
for (int  i = 0;  i < m;  ++i)
{
  double[]  output_data = compensatedData[i];
  for (int  j = 0;  j < m;  ++j)
  {
    double[]  input_data = rawData[j];
    double  coefficient = compensationMatrix[i][j];
    for (int  k = 0;  k < n;  ++k)
      output_data[k] += input_data[k] * coefficient;
    ++progress;
    SwingUtilities.invokeLater( progress_made );
  }
}

newData = new int[m][n];
for (int  i = 0;  i < m;  ++i)
{
  FCSParameter  oldp = fcs.getParameter( sensorNames[i] );
  FCSParameter  newp = fcs_data.addParameter();
  newp.setAttribute( "$P", "N", "["+sensorNames[i]+"]" );
  newp.setAttribute( "$P", "S", oldp.getAttribute( "$P", "S" ) );
  newp.setBits( oldp.getBits() );
  newp.setRange( oldp.getRange() );
  newp.setDecades( oldp.getDecades(), oldp.getMinimum() );
  newp.setGain( oldp.getGain() );

  double[]  input_data = compensatedData[i];
  int[]  transformed_data = newData[i];
  if (newp.isLog())
  {
    double[]  lookup = logicle.transform( i, newp );
    newp.setDecades( logicle.decades, logicle.minimum );
    for (int  j = 0;  j < n;  ++j)
    {
      int  k = Arrays.binarySearch( lookup, input_data[j] );
      if (k < 0)
        transformed_data[j] = -k-1;
      else
        transformed_data[j] = k;
    }
  }
  else
  {
    double  g = newp.getGain();
    for (int  j = 0;  j < n;  ++i)
      transformed_data[j] = (int) Math.round( input_data[j] * g );
  }
  ++progress;
  SwingUtilities.invokeLater( progress_made );
}

  compensatedData = null;
  SwingUtilities.invokeLater( xfrm_complete );
}
catch (Exception ex)
{
  ex.printStackTrace();
}
```

```
  }
};

Runnable  progress_made = new Runnable ()
{
  public void run ()
  {
    progressBar.setValue( progress );
  }
};

Runnable  xfrm_complete = new Runnable ()
{
  public void  run ()
  {
    progressBar.setMinimum( 0 );
    progressBar.setValue( 0 );
    progressBar.setMaximum( fcs_data.getTotal() );
    progressBar.setEnabled( true );
    statusBar.setText( fcs_data.getFile().getName() );

    new Thread( save_thread, "FCS writer" ).start();
    fcs_timer.start();
  }
};

Runnable  save_thread = new Runnable()
{
  public void run ()
  {
    Thread.currentThread().setPriority( Thread.MIN_PRIORITY );
    fcs_exception = null;
    try
    {
      fcs_handler = fcs_data.getOutputIterator();
      for (int  i = 0;  fcs_handler.hasMoreEvents();  ++i)
      {
        for (int  j = 0;  j < originalData.length;  ++j)
          fcs_handler.writeValue( originalData[j][i] );
        for (int  j = 0;  j < newData.length;  ++j)
          fcs_handler.writeValue( newData[j][i] );
      }
      fcs_handler.close();
    }
    catch (IOException ex)
    {
      ex.printStackTrace();
      fcs_exception = ex;
    }
    SwingUtilities.invokeLater( save_complete );
  }
};

Runnable  save_complete = new Runnable ()
{
  public void run ()
  {
    fcs_timer.stop();
    if (fcs_exception != null)
    {
      errorLabel1.setText( fcs_exception.getMessage() );
      errorLabel2.setText( fcs_data.getFile().getPath() );
      JOptionPane.showMessageDialog( ConverterFrame.this, errorMessage,
```

```
                    "FCS Error", JOptionPane.ERROR_MESSAGE );
      }
      make_ready();
    }
  };
  private JMenu jMenu1 = new JMenu();
  private JMenuItem jMenuItem1 = new JMenuItem();

  void function_action(ActionEvent e)
  {
    if (spectrumTable.isEditing())
      spectrumTable.getCellEditor().stopCellEditing();

    for (int  i = 0;  i < sensorNames.length;  ++i)
      for (int  j = 0;  j < this.N_PARAMETERS;  ++j)
        modelParameters.setValueAt( null, sensorNames.length + j, i );
  }

  void nomogram_action()
  {
    int columnIndex = spectrumTable.getSelectedColumn();
    if (columnIndex < 0 || fcs == null)
      return;
    FCSParameter  p;
    try
    {
      p = fcs.getParameter( sensorNames[columnIndex] );
    }
    catch (IOException ex)
    {
      return;
    }

    if (p.isLog())
    {
      LogicleFunction  function = logicleFunctions[
functionChoice.getSelectedIndex() ];
      double[]  right = function.transform( columnIndex, p );
      double[]  left = new double[ p.getRange() ];
      left[0] = function.minimum;
      interpolate( left, 0, left.length, function.maximum );
      axisDialog.nomoRelative = new double[axisDialog.nomoLabel.length][2];
      for (int  i = 0;  i < axisDialog.nomoLabel.length;  ++i)
      {
        double  x = Double.parseDouble( axisDialog.nomoLabel[i] );
        int  k = Arrays.binarySearch( left, x );
        if (k < 0)
          axisDialog.nomoRelative[i][0] = (double) (-k-1) / (double) left.length;
        else
          axisDialog.nomoRelative[i][0] = (double) k / (double) left.length;

        k = Arrays.binarySearch( right, x );
        if (k < 0)
          axisDialog.nomoRelative[i][1] = (double) (-k-1) / (double) right.length;
        else
          axisDialog.nomoRelative[i][1] = (double) k / (double) right.length;
      }
      axisDialog.repaint();
    }
    else
    {
      double  g = p.getGain();
    }
```

```
  }

  void help_action(ActionEvent e)
  {
    helpDialog.setLocationRelativeTo( this );
    helpDialog.setVisible( true );
  }

  void help_dismiss(ActionEvent e)
  {
    helpDialog.setVisible( false );
  }

  void about_action(ActionEvent e)
  {
    JOptionPane.showMessageDialog( this, aboutMessage, "About Logicle",
JOptionPane.PLAIN_MESSAGE );
  }
  public void tableChanged(TableModelEvent e)
  {
    /**@todo Implement this javax.swing.event.TableModelListener method*/
    throw new java.lang.UnsupportedOperationException("Method tableChanged() not yet
implemented.");
  }

  void scale_action(ActionEvent e)
  {
    axisDialog.setVisible( true );
    axisDialog.toFront();
  }

}
```

```
[PropertyInfo]
borderLayout1,BorderLayout,false,false, , ,false,<default>
image1,ImageIcon,false,false, , ,false,<default>
image2,ImageIcon,false,false, , ,false,<default>
image3,ImageIcon,false,false, , ,false,<default>
jButton1,JButton,false,false, , ,false,<default>
jButton2,JButton,false,false, , ,false,<default>
jButton3,JButton,false,false, , ,false,<default>
jMenuBar1,JMenuBar,false,false, , ,false,<default>
jMenuFile,JMenu,false,false, , ,false,<default>
jMenuFileExit,JMenuItem,false,false, , ,false,<default>
jMenuHelp,JMenu,false,false, , ,false,<default>
jMenuHelpAbout,JMenuItem,false,false, , ,false,<default>
jMenuItem1,JMenuItem,false,false, , ,false,<default>
jMenuItem2,JMenuItem,false,false, , ,false,<default>
jScrollPane1,JScrollPane,false,false, , ,false,<default>
jToolBar,JToolBar,false,false, , ,false,<default>
menuSpectrum,JMenuItem,false,false, , ,false,<default>
spectrum_table,JTable,false,false, , ,false,<default>
spectrumChooser,JFileChooser,false,false, , ,false,<default>
statusBar,JLabel,false,false, , ,false,<default>
[IconNames]
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame extends JFrame {
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton jButton1 = new JButton();
  private JButton jButton2 = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private JLabel statusBar = new JLabel();
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrum_table = new JTable();
  private JMenuItem menuSpectrum = new JMenuItem();
  private JFileChooser  spectrumChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );

  //Construct the frame
  public ConverterFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  //Component initialization
  private void jbInit() throws Exception  {
    image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
```

```
    image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
    image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Log Like Data Converter");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
      }
    });
    jButton1.setIcon(image1);
    jButton1.setToolTipText("Open File");
    jButton2.setIcon(image2);
    jButton2.setToolTipText("Close File");
    jButton3.setIcon(image3);
    jButton3.setToolTipText("Help");
    menuSpectrum.setText("Spectrum");
    menuSpectrum.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        spectrum_action(e);
      }
    });
    jMenuItem1.setText("Data");
    jMenuItem1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    jMenuItem2.setText("Save");
    jToolBar.add(jButton1);
    jToolBar.add(jButton2);
    jToolBar.add(jButton3);
    jMenuFile.add(menuSpectrum);
    jMenuFile.add(jMenuItem1);
    jMenuFile.add(jMenuItem2);
    jMenuFile.add(jMenuFileExit);
    jMenuHelp.add(jMenuHelpAbout);
    jMenuBar1.add(jMenuFile);
    jMenuBar1.add(jMenuHelp);
    this.setJMenuBar(jMenuBar1);
    contentPane.add(jToolBar,  BorderLayout.NORTH);
    contentPane.add(statusBar, BorderLayout.SOUTH);
    contentPane.add(jScrollPane1, BorderLayout.CENTER);
    jScrollPane1.getViewport().add(spectrum_table, null);
```

```
  }
  //File | Exit action performed
  public void jMenuFileExit_actionPerformed(ActionEvent e) {
    System.exit(0);
  }
  //Help | About action performed
  public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
  }
  //Overridden so we can exit when window is closed
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      jMenuFileExit_actionPerformed(null);
    }
  }

  private static final String[]  stringArray = new String[0];
  private JMenuItem jMenuItem1 = new JMenuItem();
  private JMenuItem jMenuItem2 = new JMenuItem();

  void spectrum_action (
      ActionEvent e )
  {
    int returnVal = spectrumChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        BufferedReader  br = new BufferedReader( new FileReader(
spectrumChooser.getSelectedFile() ) );
        br.readLine();
        br.readLine();
        StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
        ArrayList  al = new ArrayList();
        while (st.hasMoreTokens())
          al.add( st.nextToken() );
        String[]  columns = (String[]) al.toArray( stringArray );
        Double[][]  rows = new Double[columns.length][];
        for (int  r = 0;  r < rows.length;  ++r)
        {
          Double[]  row = new Double[columns.length];
          rows[r] = row;
          st = new StringTokenizer( br.readLine(), "\t" );
          for (int  c = 0;  c < columns.length;  ++c)
          {
            row[c] = new Double( Double.parseDouble( st.nextToken() ) );
          }
        }
        spectrum_table.setModel( new DefaultTableModel( rows, columns ) );
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }

  void data_action(ActionEvent e)
  {
    int returnVal = spectrumChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
```

```
      {
        FCSFile  fcs = new FCSFile( spectrumChooser.getSelectedFile() );
        for (Iterator  it = fcs.getKeywords().entrySet().iterator();  it.hasNext();
)
        {
          Map.Entry  entry = (Map.Entry) it.next();
          System.out.print( entry.getKey() );
          System.out.print( " = " );
          System.out.println( entry.getValue() );
        }
        int[][]  event = new int[fcs.getTotal()][];
        ListModeIterator  lmi = fcs.getListModeIterator();
        for (int  i = 0;  lmi.hasNext();  ++i)
          event[i] = (int[]) lmi.next().clone();
        int  total = fcs.getTotal();
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }
}
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame extends JFrame {
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton jButton1 = new JButton();
  private JButton jButton2 = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private JLabel statusBar = new JLabel();
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrum_table = new JTable();
  private JMenuItem menuSpectrum = new JMenuItem();
  private JFileChooser  spectrumChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );
  private JMenuItem jMenuItem1 = new JMenuItem();
  private JMenuItem jMenuItem2 = new JMenuItem();

  //Construct the frame
  public ConverterFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  //Component initialization
  private void jbInit() throws Exception  {
    image1 = new
```

```
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
    image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
    image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Log Like Data Converter");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
      }
    });
    jButton1.setIcon(image1);
    jButton1.setToolTipText("Open File");
    jButton2.setIcon(image2);
    jButton2.setToolTipText("Close File");
    jButton3.setIcon(image3);
    jButton3.setToolTipText("Help");
    menuSpectrum.setText("Spectrum");
    menuSpectrum.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        spectrum_action(e);
      }
    });
    jMenuItem1.setText("Data");
    jMenuItem1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    jMenuItem2.setText("Save");
    jToolBar.add(jButton1);
    jToolBar.add(jButton2);
    jToolBar.add(jButton3);
    jMenuFile.add(menuSpectrum);
    jMenuFile.add(jMenuItem1);
    jMenuFile.add(jMenuItem2);
    jMenuFile.add(jMenuFileExit);
    jMenuHelp.add(jMenuHelpAbout);
    jMenuBar1.add(jMenuFile);
    jMenuBar1.add(jMenuHelp);
    this.setJMenuBar(jMenuBar1);
    contentPane.add(jToolBar,  BorderLayout.NORTH);
    contentPane.add(statusBar, BorderLayout.SOUTH);
```

```
    contentPane.add(jScrollPane1, BorderLayout.CENTER);
    jScrollPane1.getViewport().add(spectrum_table, null);
  }
  //File | Exit action performed
  public void jMenuFileExit_actionPerformed(ActionEvent e) {
    System.exit(0);
  }
  //Help | About action performed
  public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
  }
  //Overridden so we can exit when window is closed
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      jMenuFileExit_actionPerformed(null);
    }
  }

  private static final String[]  stringArray = new String[0];
  private String[]  sensor_names;

  void spectrum_action (
      ActionEvent e )
  {
    int returnVal = spectrumChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        BufferedReader  br = new BufferedReader( new FileReader(
spectrumChooser.getSelectedFile() ) );
        br.readLine();
        br.readLine();
        StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
        ArrayList  al = new ArrayList();
        while (st.hasMoreTokens())
          al.add( st.nextToken() );
        String[]  columns = sensor_names = (String[]) al.toArray( stringArray );
        Double[][]  rows = new Double[columns.length][];
        for (int  r = 0;  r < rows.length;  ++r)
        {
          Double[]  row = new Double[columns.length];
          rows[r] = row;
          st = new StringTokenizer( br.readLine(), "\t" );
          for (int  c = 0;  c < columns.length;  ++c)
          {
            row[c] = new Double( Double.parseDouble( st.nextToken() ) );
          }
        }
        spectrum_table.setModel( new DefaultTableModel( rows, columns ) );
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }

  private void  interpolate (
      double[]  func,
      int  i,
      int  n,
      double  x )
```

```
  {
    int  m = n / 2;
    double  y = Math.sqrt( func[i] * x );

    func[ i + m ] = y;

    if (m > 1 )
    {
      interpolate( func, i, m, y );
      interpolate( func, i + m, m, x );
    }

  }

  void data_action(ActionEvent e)
  {
    int returnVal = spectrumChooser.showOpenDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        FCSFile  fcs = new FCSFile( spectrumChooser.getSelectedFile() );
        for (Iterator  it = fcs.getKeywords().entrySet().iterator();  it.hasNext();
)
        {
          Map.Entry  entry = (Map.Entry) it.next();
          System.out.print( entry.getKey() );
          System.out.print( " = " );
          System.out.println( entry.getValue() );
        }
        int  n = fcs.getTotal();
        int  m = fcs.getParameters();
        int[][]  int_value = new int[m][n];
        ListModeIterator  lmi = fcs.getListModeIterator();
        for (int  i = 0;  lmi.hasMoreEvents();  ++i)
          for (int  j = 0;  lmi.hasMoreValues();  ++j)
            int_value[j][i] = lmi.nextValue();
        for (int  j = 1;  j <= m;  ++j)
        {
          FCSParameter  p = fcs.getParameter( j );
          if (p.isLog())
          {
            double[]  antilog = new double[ p.getRange() ];
            antilog[0] = p.getMinimum();
            interpolate( antilog, 0, p.getRange(), p.getMaximum() );
            for (int  i = 0;  i < n;  ++i)
            {
              int  int_in = int_value[j][i];
              double  value = antilog[ int_in ];
              int  int_out = java.util.Arrays.binarySearch( antilog, value );
              System.out.print( int_in );
              System.out.print( " -> " );
              System.out.print( value );
              System.out.print( " -> " );
              System.out.println( int_out );
              if (int_out < 0)
                int_out = -int_out;
            }
          }
        }
      }
      catch (Exception ex)
      {
```

```
            ex.printStackTrace();
        }
    }
}
}
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;
import java.util.StringTokenizer;
import javax.swing.*;
import javax.swing.table.*;

import org.isac.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame extends JFrame {
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton jButton1 = new JButton();
  private JButton jButton2 = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private JLabel statusBar = new JLabel();
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrum_table = new JTable();
  private JMenuItem menuSpectrum = new JMenuItem();
  private JFileChooser  spectrumChooser = new JFileChooser( System.getProperty(
"user.home", "" ) );
  private JMenuItem jMenuItem1 = new JMenuItem();
  private JMenuItem jMenuItem2 = new JMenuItem();

  //Construct the frame
  public ConverterFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
  //Component initialization
  private void jbInit() throws Exception  {
```

```
    image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
    image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
    image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
    contentPane = (JPanel) this.getContentPane();
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Log Like Data Converter");
    statusBar.setText(" ");
    jMenuFile.setText("File");
    jMenuFileExit.setText("Exit");
    jMenuFileExit.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuFileExit_actionPerformed(e);
      }
    });
    jMenuHelp.setText("Help");
    jMenuHelpAbout.setText("About");
    jMenuHelpAbout.addActionListener(new ActionListener()  {
      public void actionPerformed(ActionEvent e) {
        jMenuHelpAbout_actionPerformed(e);
      }
    });
    jButton1.setIcon(image1);
    jButton1.setToolTipText("Open File");
    jButton2.setIcon(image2);
    jButton2.setToolTipText("Close File");
    jButton3.setIcon(image3);
    jButton3.setToolTipText("Help");
    menuSpectrum.setText("Spectrum");
    menuSpectrum.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        spectrum_action(e);
      }
    });
    jMenuItem1.setText("Data");
    jMenuItem1.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        data_action(e);
      }
    });
    jMenuItem2.setText("Save");
    jToolBar.add(jButton1);
    jToolBar.add(jButton2);
    jToolBar.add(jButton3);
    jMenuFile.add(menuSpectrum);
    jMenuFile.add(jMenuItem1);
    jMenuFile.add(jMenuItem2);
    jMenuFile.add(jMenuFileExit);
    jMenuHelp.add(jMenuHelpAbout);
    jMenuBar1.add(jMenuFile);
    jMenuBar1.add(jMenuHelp);
    this.setJMenuBar(jMenuBar1);
    contentPane.add(jToolBar,  BorderLayout.NORTH);
```

```
  contentPane.add(statusBar, BorderLayout.SOUTH);
  contentPane.add(jScrollPane1, BorderLayout.CENTER);
  jScrollPane1.getViewport().add(spectrum_table, null);
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
}
//Help | About action performed
public void jMenuHelpAbout_actionPerformed(ActionEvent e) {
}
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
  }
}

private static final String[]  stringArray = new String[0];
private FCSFile  fcs;
private String[]  sensor_names;
private int[][]  int_data;
private double[][]  raw_data;
private double[][]  compensated_data;
private double[][]  compensation_matrix;

void  compensate ()
throws IOException
{
  int  n = fcs.getTotal();
  int  m = sensor_names.length;

  raw_data = new double[ m ][ n ];
  compensated_data = new double[ m ][ n ];
  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensor_names[i] );
    int[]  input_data = int_data[p.getIndex()-1];
    double[]  output_data = raw_data[i];
    if (p.isLog())
    {
      double[]  antilog = new double[ p.getRange() ];
      antilog[0] = p.getMinimum();
      interpolate( antilog, 0, p.getRange(), p.getMaximum() );
      for (int  j = 0;  j < n;  ++i)
        output_data[j] = antilog[ input_data[j] ];
    }
    else
    {
      double  g = p.getGain();
      for (int  j = 0;  j < n;  ++i)
        output_data[j] = g * input_data[j];
    }
  }

  for (int  i = 0;  i < m;  ++i)
  {
    double[]  mult = new double[m];
    mult[i] = 1;

    double[]  output_data = compensated_data[i];
    for (int  j = 0;  j < n;  ++j)
```

```
    {
      double  x = 0;
      for (int  k = 0; k < m; ++k)
        x += raw_data[k][j] * mult[k];
      output_data[j] = x;
    }
  }

  for (int  i = 0;  i < m;  ++i)
  {
    FCSParameter  p = fcs.getParameter( sensor_names[i] );
    double[]  input_data = compensated_data[i];
    int[]  transformed_data = new int[n];
    if (p.isLog())
    {
      double[]  lookup = new double[ p.getRange() ];
      lookup[0] = p.getMinimum();
      interpolate( lookup, 0, p.getRange(), p.getMaximum() );
      for (int  j = 0;  j < n;  ++i)
      {
        int  k = Arrays.binarySearch( lookup, input_data[j] );
        if (k < 0)
          transformed_data[j] = -k;
        else
          transformed_data[j] = k;
      }
    }
    else
    {
      double  g = p.getGain();
      for (int  j = 0;  j < n;  ++i)
        transformed_data[j] = (int) Math.floor( input_data[j] / g );
    }
  }
}

void spectrum_action (
    ActionEvent e )
{
  int returnVal = spectrumChooser.showOpenDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      BufferedReader  br = new BufferedReader( new FileReader(
spectrumChooser.getSelectedFile() ) );
      br.readLine();
      br.readLine();
      StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
      ArrayList  al = new ArrayList();
      while (st.hasMoreTokens())
        al.add( st.nextToken() );
      String[]  columns = sensor_names = (String[]) al.toArray( stringArray );
      Double[][]  rows = new Double[columns.length][];
      for (int  r = 0;  r < rows.length;  ++r)
      {
        Double[]  row = new Double[columns.length];
        rows[r] = row;
        st = new StringTokenizer( br.readLine(), "\t" );
        for (int  c = 0;  c < columns.length;  ++c)
        {
          row[c] = new Double( Double.parseDouble( st.nextToken() ) );
        }
```

```
      }
      spectrum_table.setModel( new DefaultTableModel( rows, columns ) );
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }

  compensation_matrix = new double[ sensor_names.length ][ sensor_names.length ];
  for (int  i = 0;  i < sensor_names.length;  ++i)
    compensation_matrix[i][i] = 1.0;
}

private void  interpolate (
    double[]  func,
    int  i,
    int  n,
    double  x )
{
  int  m = n / 2;
  double  y = Math.sqrt( func[i] * x );

  func[ i + m ] = y;

  if (m > 1 )
  {
    interpolate( func, i, m, y );
    interpolate( func, i + m, m, x );
  }

}

void data_action(ActionEvent e)
{
  int returnVal = spectrumChooser.showOpenDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      FCSFile  fcs = new FCSFile( spectrumChooser.getSelectedFile() );
      for (Iterator  it = fcs.getKeywords().entrySet().iterator();  it.hasNext();
)
      {
        Map.Entry  entry = (Map.Entry) it.next();
        System.out.print( entry.getKey() );
        System.out.print( " = " );
        System.out.println( entry.getValue() );
      }
      int  n = fcs.getTotal();
      int  m = fcs.getParameters();
      int[][]  int_value = new int[m][n];
      ListModeIterator  lmi = fcs.getListModeIterator();
      for (int  i = 0;  lmi.hasMoreEvents();  ++i)
        for (int  j = 0;  lmi.hasMoreValues();  ++j)
          int_value[j][i] = lmi.nextValue();
      for (int  j = 1;  j <= m;  ++j)
      {
        int[]  ia = int_value[ j-1 ];
        FCSParameter  p = fcs.getParameter( j );
        if (p.isLog())
        {
          double[]  antilog = new double[ p.getRange() ];
```

```
            antilog[0] = p.getMinimum();
            interpolate( antilog, 0, p.getRange(), p.getMaximum() );
            for (int  i = 0;  i < n;  ++i)
            {
              int  int_in = ia[i];
              double  value = antilog[ int_in ];
              int  int_out = java.util.Arrays.binarySearch( antilog, value );
              System.out.print( int_in );
              System.out.print( " -> " );
              System.out.print( value );
              System.out.print( " -> " );
              System.out.println( int_out );
              if (int_out < 0)
                int_out = -int_out;
            }
          }
        }
        this.fcs = fcs;
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }
}
```

```
package edu.stanford.facs.loglike;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

import javax.swing.*;
import javax.swing.Timer;
import javax.swing.border.*;
import javax.swing.event.*;
import javax.swing.table.*;

import org.isac.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class ConverterFrame
extends JFrame implements TableModelListener
{
  private JPanel contentPane;
  private JMenuBar jMenuBar1 = new JMenuBar();
  private JMenu jMenuFile = new JMenu();
  private JMenuItem jMenuFileExit = new JMenuItem();
  private JMenu jMenuHelp = new JMenu();
  private JMenuItem jMenuHelpAbout = new JMenuItem();
  private JToolBar jToolBar = new JToolBar();
  private JButton openButton = new JButton();
  private JButton saveButton = new JButton();
  private JButton jButton3 = new JButton();
  private ImageIcon image1;
  private ImageIcon image2;
  private ImageIcon image3;
  private BorderLayout borderLayout1 = new BorderLayout();
  private JScrollPane jScrollPane1 = new JScrollPane();
  private JTable spectrumTable = new JTable();
  private JMenuItem spectrumMenuItem = new JMenuItem();
  private JFileChooser  fileChooser = new JFileChooser( System.getProperty(
 "user.home", "" ) );
  private JMenuItem openMenuItem = new JMenuItem();
  private JMenuItem saveMenuItem = new JMenuItem();
  private JButton spectrumButton = new JButton();
  private JComboBox functionChoice = new JComboBox();
  private JLabel jLabel1 = new JLabel();
  private JDialog helpDialog = new JDialog();
  private JPanel jPanel1 = new JPanel();
  private JButton helpDismiss = new JButton();
  private JScrollPane jScrollPane2 = new JScrollPane();
  private JEditorPane helpPane = new JEditorPane();
  private JPanel aboutMessage = new JPanel();
  private JLabel jLabel2 = new JLabel();
  private JLabel jLabel3 = new JLabel();
  private GridLayout gridLayout1 = new GridLayout();
  private JLabel jLabel4 = new JLabel();
  private Component component2;
  private JLabel jLabel5 = new JLabel();
```

```
  private JLabel jLabel6 = new JLabel();
  private Border border1;
  private Component component3;
  private JLabel jLabel7 = new JLabel();
  private JPanel jPanel2 = new JPanel();
  private JLabel statusBar = new JLabel();
  private JProgressBar progressBar = new JProgressBar();
  private BorderLayout borderLayout3 = new BorderLayout();
  private BorderLayout borderLayout2 = new BorderLayout();
  private JMenuItem jMenuItem2 = new JMenuItem();
  private JLabel jLabel8 = new JLabel();
  private Component component1;
  private Component component4;
  private JPanel errorMessage = new JPanel();
  private GridLayout gridLayout2 = new GridLayout();
  private JLabel errorLabel1 = new JLabel();
  private JLabel errorLabel2 = new JLabel();
  private JMenu menuScale = new JMenu();
  private JMenuItem jMenuItem1 = new JMenuItem();
  private JMenuItem jMenuItem3 = new JMenuItem();
  private JMenu jMenu3 = new JMenu();
  private JMenuItem jMenuItem4 = new JMenuItem();
  private JMenuItem jMenuItem5 = new JMenuItem();
  private AxisDialog  axisDialog;

  //Construct the frame
  public ConverterFrame()
  {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try
    {
      jbInit();
      spectrumTable.setModel( modelParameters );
      modelParameters.addTableModelListener( new TableModelListener ()
          {
        public void tableChanged(TableModelEvent e)
        {
          if (spectrumTable.getSelectedColumn() >= 0 && e.getColumn() ==
spectrumTable.getSelectedColumn())
            nomogram_action();
        }
          } );
      spectrumTable.getTableHeader().setReorderingAllowed( false );
      spectrumTable.setColumnSelectionAllowed( true );
      spectrumTable.setRowSelectionAllowed( false );
      spectrumTable.getColumnModel().getSelectionModel().setSelectionMode(
ListSelectionModel.SINGLE_SELECTION );
      spectrumTable.getColumnModel().addColumnModelListener(
      new TableColumnModelListener ()
      {
        public void columnSelectionChanged( ListSelectionEvent  lse )
        {
          nomogram_action();
        }
        public void columnMarginChanged( ChangeEvent  ce )
        {
        }
        public void columnMoved( TableColumnModelEvent  tcme )
        {
        }
        public void columnRemoved( TableColumnModelEvent  tcme )
        {
        }
```

```
      public void columnAdded( TableColumnModelEvent  tcme )
      {
      }
    } );
    axisDialog = new AxisDialog( this, "Logicle Scale", false );
    axisDialog.pack();
    helpPane.setPage( ConverterFrame.class.getResource( "help.html" ) );
    helpDialog.pack();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}
//Component initialization
private void jbInit() throws Exception  {
  image1 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("openFile.gif")
);
  image2 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("closeFile.gif"
));
  image3 = new
ImageIcon(edu.stanford.facs.loglike.ConverterFrame.class.getResource("help.gif"));

//setIconImage(Toolkit.getDefaultToolkit().createImage(ConverterFrame.class.getResou
rce("[Your Icon]")));
  contentPane = (JPanel) this.getContentPane();
  component2 = Box.createVerticalStrut(8);
  border1 = BorderFactory.createEmptyBorder(10,15,10,25);
  component3 = Box.createVerticalStrut(8);
  component1 = Box.createVerticalStrut(8);
  component4 = Box.createVerticalStrut(8);
  contentPane.setLayout(borderLayout1);
  this.setSize(new Dimension(400, 300));
  this.setTitle("Logicle Fluorescence Compensation");
  jMenuFile.setText("File");
  jMenuFileExit.setText("Exit");
  jMenuFileExit.addActionListener(new ActionListener()  {
    public void actionPerformed(ActionEvent e) {
      jMenuFileExit_actionPerformed(e);
    }
  });
  jMenuHelp.setText("Help");
  jMenuHelpAbout.setText("About ...");
  jMenuHelpAbout.addActionListener(new ActionListener()  {
    public void actionPerformed(ActionEvent e) {
      about_action(e);
    }
  });
  openButton.setIcon(image1);
  openButton.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      data_action(e);
    }
  });
  openButton.setEnabled(false);
  openButton.setToolTipText("Open FCS File");
  saveButton.setIcon(image2);
  saveButton.addActionListener(new java.awt.event.ActionListener()
  {
```

```
    public void actionPerformed(ActionEvent e)
    {
      save_action(e);
    }
  });
  saveButton.setEnabled(false);
  saveButton.setToolTipText("Save Extended FCS File");
  jButton3.setIcon(image3);
  jButton3.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      help_action(e);
    }
  });
  jButton3.setToolTipText("Help");
  spectrumMenuItem.setIcon(new
ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
  spectrumMenuItem.setText("Spectrum ...");
  spectrumMenuItem.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      spectrum_action(e);
    }
  });
  openMenuItem.setEnabled(false);
  openMenuItem.setIcon(image1);
  openMenuItem.setText("Data ...");
  openMenuItem.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      data_action(e);
    }
  });
  saveMenuItem.setEnabled(false);
  saveMenuItem.setIcon(image2);
  saveMenuItem.setText("Save As ...");
  saveMenuItem.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      save_action(e);
    }
  });
  functionChoice.setFont(new java.awt.Font("Dialog", 1, 14));
  functionChoice.setMaximumSize(new Dimension(32767, 27));
  functionChoice.setMinimumSize(new Dimension(138, 27));
  functionChoice.setPreferredSize(new Dimension(142, 27));
  functionChoice.setToolTipText("Select a Data Transformation");
  functionChoice.setModel(functionModel);
  functionChoice.addActionListener(new java.awt.event.ActionListener()
  {
    public void actionPerformed(ActionEvent e)
    {
      function_action(e);
    }
  });
  spectrumButton.setMaximumSize(new Dimension(51, 27));
  spectrumButton.setMinimumSize(new Dimension(51, 27));
  spectrumButton.setToolTipText("Load Spectrum Matrix");
  spectrumButton.setIcon(new
ImageIcon(ConverterFrame.class.getResource("spectrum.jpg")));
  spectrumButton.addActionListener(new java.awt.event.ActionListener()
```

```
    {
      public void actionPerformed(ActionEvent e)
      {
        spectrum_action(e);
      }
    });
    jLabel1.setPreferredSize(new Dimension(17, 17));
    spectrumTable.setRowSelectionAllowed(false);
    helpDismiss.setText("Close");
    helpDismiss.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        help_dismiss(e);
      }
    });
    helpDialog.setTitle("Logicle Help");
    helpDialog.getContentPane().setLayout(borderLayout2);
    jMenuItem2.setText("Help ...");
    jMenuItem2.addActionListener(new java.awt.event.ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        help_action(e);
      }
    });
    helpPane.setPreferredSize(new Dimension(400, 400));
    helpPane.setEditable(false);
    helpPane.setText("<html>\r\n  <head>\r\n\r\n  </head>\r\n  <body>\r\n    <p>\r\n
Logicle Data Converter " +
    "    \r\n    </p>\r\n  </body>\r\n</html>\r\n");
    helpPane.setContentType("text/html");
    jScrollPane2.setPreferredSize(new Dimension(400, 400));
    jLabel2.setFont(new java.awt.Font("Dialog", 1, 14));
    jLabel2.setAlignmentY((float) 0.0);
    jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel2.setHorizontalTextPosition(SwingConstants.LEADING);
    jLabel2.setIconTextGap(0);
    jLabel2.setText("Logicle Fluorescence Compensation");
    jLabel3.setAlignmentX((float) 0.5);
    jLabel3.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel3.setIconTextGap(0);
    jLabel3.setText("Wayne A. Moore");
    aboutMessage.setLayout(gridLayout1);
    gridLayout1.setColumns(1);
    gridLayout1.setRows(11);
    jLabel4.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel4.setIconTextGap(0);
    jLabel4.setText("January 2003");
    jLabel5.setFont(new java.awt.Font("Dialog", 2, 10));
    jLabel5.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel5.setHorizontalTextPosition(SwingConstants.LEADING);
    jLabel5.setIconTextGap(0);
    jLabel5.setText("Copyright 2002, 2003, by The Board of Trustees");
    jLabel6.setFont(new java.awt.Font("Dialog", 2, 10));
    jLabel6.setHorizontalAlignment(SwingConstants.CENTER);
    jLabel6.setHorizontalTextPosition(SwingConstants.LEADING);
    jLabel6.setIconTextGap(0);
    jLabel6.setText("of the Leland Stanford Jr. University");
    aboutMessage.setBorder(border1);
    aboutMessage.setMinimumSize(new Dimension(60, 40));
    aboutMessage.setPreferredSize(new Dimension(300, 250));
    jLabel7.setHorizontalAlignment(SwingConstants.CENTER);
```

```
jLabel7.setText("David R. Parks");
statusBar.setText(" ");
jPanel2.setLayout(borderLayout3);
progressBar.setString("Compensate");
jLabel8.setHorizontalAlignment(SwingConstants.CENTER);
jLabel8.setText("Version 1.1");
errorMessage.setLayout(gridLayout2);
gridLayout2.setColumns(1);
gridLayout2.setRows(0);
errorLabel1.setHorizontalAlignment(SwingConstants.CENTER);
errorLabel2.setHorizontalAlignment(SwingConstants.CENTER);
errorLabel2.setText("jLabel10");
menuScale.setEnabled(false);
menuScale.setText("Scale");
jMenuItem1.setText("View");
jMenuItem1.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    view_action(e);
  }
});
jMenuItem3.setText("Save");
jMenuItem3.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    new_scale_action(e);
  }
});
jMenuItem4.setText("1st %tile");
jMenuItem4.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    automatic_action_1_neg(e);
  }
});
jMenu3.setText("Automatic");
jMenuItem5.setText("5th %tile");
jMenuItem5.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    automatic_action_5_neg(e);
  }
});
jMenuItem6.setText("Minimum");
jMenuItem6.addActionListener(new java.awt.event.ActionListener()
{
  public void actionPerformed(ActionEvent e)
  {
    automatic_action_0_neg(e);
  }
});
jToolBar.add(spectrumButton, null);
jToolBar.add(openButton);
jToolBar.add(saveButton);
jToolBar.add(jButton3);
jToolBar.add(jLabel1, null);
jToolBar.add(functionChoice, null);
jMenuFile.add(spectrumMenuItem);
jMenuFile.add(openMenuItem);
```

```
jMenuFile.add(saveMenuItem);
jMenuFile.addSeparator();
jMenuFile.add(jMenuFileExit);
jMenuHelp.add(jMenuItem2);
jMenuHelp.addSeparator();
jMenuHelp.add(jMenuHelpAbout);
jMenuBar1.add(jMenuFile);
jMenuBar1.add(menuScale);
jMenuBar1.add(jMenuHelp);
this.setJMenuBar(jMenuBar1);
contentPane.add(jToolBar,  BorderLayout.NORTH);
contentPane.add(jScrollPane1, BorderLayout.CENTER);
contentPane.add(jPanel2, BorderLayout.SOUTH);
jPanel2.add(statusBar,  BorderLayout.CENTER);
jPanel2.add(progressBar,  BorderLayout.EAST);
jScrollPane1.getViewport().add(spectrumTable, null);
helpDialog.getContentPane().add(jPanel1, BorderLayout.SOUTH);
jPanel1.add(helpDismiss, null);
helpDialog.getContentPane().add(jScrollPane2, BorderLayout.CENTER);
jScrollPane2.getViewport().add(helpPane, null);
aboutMessage.add(jLabel2, null);
aboutMessage.add(component4, null);
aboutMessage.add(jLabel8, null);
aboutMessage.add(component1, null);
aboutMessage.add(jLabel3, null);
aboutMessage.add(jLabel7, null);
aboutMessage.add(jLabel4, null);
aboutMessage.add(component2, null);
aboutMessage.add(jLabel5, null);
aboutMessage.add(jLabel6, null);
aboutMessage.add(component3, null);
errorMessage.add(errorLabel1, null);
errorMessage.add(errorLabel2, null);
menuScale.add(jMenuItem1);
menuScale.add(jMenuItem3);
menuScale.add(jMenu3);
jMenu3.add(jMenuItem6);
jMenu3.add(jMenuItem4);
jMenu3.add(jMenuItem5);
}
//File | Exit action performed
public void jMenuFileExit_actionPerformed(ActionEvent e) {
  System.exit(0);
}
//Help | About action performed
//Overridden so we can exit when window is closed
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
    jMenuFileExit_actionPerformed(null);
  }
}

private static final String[]  stringArray = new String[0];
private static final int  N_PARAMETERS = 2;
private static final int FUZZY_BITS = 8;
private FCSFile  fcs;
private int  fcsTotal;
private String  spectrumName;
private String[]  sensorNames = new String[0];
private int[][]  originalData;
private int[][]  newData;
private double[][]  rawData;
```

```
private double[][]  compensatedData;
private double[][]  compensationMatrix;
private int  progress;
private LogicleFunction  logicle;
private LogicleFunction  logarithm = new LogicleFunction();

public class  LogicleFunction
{
  double  decades, minimum, maximum;

  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double[]  lookup = new double[ parameter.getRange() ];
    maximum = parameter.getMaximum();
    minimum = parameter.getMinimum();
    decades = parameter.getDecades();
    lookup[0] = minimum;
    interpolate( lookup, 0, lookup.length, maximum );

    return  lookup;
  }

  public void  automaticScale (
      int  column,
      FCSParameter  parameter,
      double  scale )
  {
    modelParameters.setParameter( scale, sensorNames.length, column );
    modelParameters.setParameter( Double.NaN, sensorNames.length + 1, column );
  }
}

public final  LogicleFunction  logLinearSplice = new LogicleFunction ()
{

  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double  linear_range = modelParameters.getParameter( sensorNames.length,
column );
    if (Double.isNaN( linear_range ) || linear_range <= 0)
      return  super.transform( column, parameter );

    maximum = parameter.getMaximum();
    double ln_range = Math.log( maximum / linear_range ) + 2;
    decades = ln_range / ln_10;
    minimum = maximum / Math.exp( ln_range );

    double[]  lookup = new double[ parameter.getRange() ];
    lookup[0] = minimum;
    interpolate( lookup, 0, lookup.length, maximum );

    int  splice = (int) Math.round( 2 * parameter.getRange() / ln_range );
    if (splice > 0 && splice < parameter.getRange() - 1)
    {
      double  delta = (lookup[splice+1] - lookup[splice-1])/2;
      double  base = lookup[splice];
      for (int  j = 0;  j < splice;  ++j)
        lookup[j] = base - (splice - j) * delta;
    }
```

```
    return  lookup;
  }
};

public final  LogicleFunction  biexponential = new LogicleFunction ()
{
  // this value should be suitable for immunofluorescence on lymphocytes
  // it may need to be modified for other types of data
  double  userDecades = 4.5;  //1.1

  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double  width = modelParameters.getParameter( sensorNames.length, column );
    if (Double.isNaN( width ))
      return  super.transform( column, parameter );

//      decades = parameter.getDecades();  //1.0
    decades = userDecades - width / 2;  //1.1

    double  extra = modelParameters.getParameter( sensorNames.length + 1, column
);
    if (Double.isNaN( extra ) || extra < 0)
      extra = 0;
    extra += width / 2;  //1.1
    int  zero = (int) Math.round( extra * parameter.getRange() / (extra + decades)
);
    if (zero > 0)
    {
      extra = zero * (decades + extra) / parameter.getRange(); // so extra falls
exactly on channel boundry
      decades = extra * parameter.getRange() / zero;
    }

//      width /= decades;  //1.0
    width /= 2 * decades;  //1.1

    maximum = parameter.getMaximum();
    double positive_range = ln_10 * decades;
    minimum = maximum / Math.exp( positive_range );

    double[]  positive = new double[ parameter.getRange() ];
    positive[0] = 1;
    interpolate( positive, 0, positive.length, Math.exp( positive_range ) );

    double negative_range = root( positive_range, width );
    double[]  negative = new double[ positive.length ];
    negative[0] = 1;
    interpolate( negative, 0, negative.length, Math.exp( -negative_range ) );

    // this is just the second translation formula
    double  s = Math.exp( (positive_range + negative_range) * (width + extra /
decades) );
    for (int  j = 0;  j < negative.length;  ++j)
      negative[j] *= s;

    s = positive[zero] - negative[zero];
    for (int  j = zero;  j < positive.length;  ++j)
      positive[j] = minimum * (positive[j] - negative[j] - s);
    // reflect around zero
    for (int  j = 0;  j < zero;  ++j)
```

```
        positive[j] = -positive[2 * zero - j];

      return  positive;
    }

    public void  automaticScale (
        int  column,
        FCSParameter  parameter,
        double  scale )
    {
//      double  minimum = parameter.getMinimum();  //1.0
      double  minimum = parameter.getMaximum() / Math.exp( userDecades * ln_10 );
//1.1
      if (scale <= 0 || scale <= minimum)
      {
        modelParameters.setParameter( 0, sensorNames.length, column );
        modelParameters.setParameter( Double.NaN, sensorNames.length + 1, column );
      }
      else
      {
//        double  decades = Math.log( scale / parameter.getMinimum() ) / ln_10;
//1.0
        double  decades = Math.log( scale / minimum ) / ln_10;  //1.1
        modelParameters.setParameter( decades, sensorNames.length, column );
//        modelParameters.setParameter( decades, sensorNames.length + 1, column );
//1.0
        modelParameters.setParameter( Double.NaN, sensorNames.length + 1, column );
//1.1
      }
    }
  };


  public final  LogicleFunction  hyperbolic = new LogicleFunction ()
  {

    public double[]  transform (
        int  column,
        FCSParameter  parameter )
    {
      double  linear_range = modelParameters.getParameter( sensorNames.length,
column );
      if (Double.isNaN( linear_range ))
        return  super.transform( column, parameter );

      maximum = parameter.getMaximum();
      double  scale_factor = linear_range / (Math.E - 1/Math.E);
      double ln_range = Math.log( maximum / scale_factor ) + 2;
      decades = ln_range / ln_10;
      minimum = maximum / Math.exp( ln_range );

      double[]  lookup = new double[ parameter.getRange() ];
      lookup[0] = 1 / Math.E;
      interpolate( lookup, 0, lookup.length, maximum / scale_factor );

      for (int  j = 0;  j < lookup.length;  ++j)
        lookup[j] = (lookup[j] - 1/lookup[j]) * scale_factor;

      return  lookup;
    }
  };

  public final  LogicleFunction  logPlus = new LogicleFunction ()
```

```
{
  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double  parameter1 = modelParameters.getParameter( sensorNames.length, column
);
    double  parameter2 = modelParameters.getParameter( sensorNames.length + 1,
column );
    if (Double.isNaN( parameter1 ))
      return  super.transform( column, parameter );
    if (Double.isNaN( parameter2 ))
      parameter2 = 0;

    double[]  lookup = super.transform( column, parameter );

//    parameter2 += parameter1 * Math.log( parameter.getMinimum() ) / ln_10;
    parameter1 *= parameter.getDecades() / parameter.getRange();
    for (int  j = 0;  j < lookup.length;  ++j)
      lookup[j] += parameter1 * j + parameter2;

    return  lookup;
  }

  public void  automaticScale (
      int  column,
      FCSParameter  parameter,
      double  scale )
  {
    modelParameters.setParameter( 0, sensorNames.length, column );
    modelParameters.setParameter( -scale, sensorNames.length + 1, column );
  }
};

public final  LogicleFunction  logPlusConstant = new LogicleFunction ()
{
  public double[]  transform (
      int  column,
      FCSParameter  parameter )
  {
    double  constant = modelParameters.getParameter( sensorNames.length, column );
    double  parameter2 = modelParameters.getParameter( sensorNames.length + 1,
column );
    if (Double.isNaN( constant ))
      return  super.transform( column, parameter );
    double[]  lookup = super.transform( column, parameter );

    for (int  j = 0;  j < lookup.length;  ++j)
      lookup[j] -= constant;

    return  lookup;
  }

  public void  automaticScale (
      int  column,
      FCSParameter  parameter,
      double  scale )
  {
    modelParameters.setParameter( scale, sensorNames.length, column );
    modelParameters.setParameter( Double.NaN, sensorNames.length + 1, column );
  }
};
```

```
private class  ModelParameters
extends AbstractTableModel
{
  private double[][]  parameters = new double[0][];

  public int getRowCount()
  {
    return  parameters.length;
  }

  public int getColumnCount()
  {
    return  sensorNames.length;
  }

  public String getColumnName (
      int columnIndex )
  {
    return  sensorNames[ columnIndex ];
  }

  public Class getColumnClass (
      int columnIndex )
  {
    return  super.getColumnClass( columnIndex );
  }

  public boolean isCellEditable (
      int rowIndex,
      int columnIndex )
  {
    return  true;
  }

  public Object getValueAt (
      int rowIndex,
      int columnIndex )
  {
    double  value = parameters[ rowIndex ][ columnIndex ];
    if (Double.isNaN( value ))
      return  null;
    else
      return  String.valueOf( value );
  }

  public void setValueAt (
      Object aValue,
      int rowIndex,
      int columnIndex)
  {
    double  dvalue;
    if (aValue instanceof Number)
      dvalue = ((Number) aValue).doubleValue();
    else if (aValue instanceof String)
    {
      try
      {
        dvalue = Double.valueOf( (String) aValue ).doubleValue();
      }
      catch (NumberFormatException ex)
      {
        dvalue = Double.NaN;
      }
```

```
    }
    else
      dvalue = Double.NaN;
    if (Double.isNaN( dvalue) && rowIndex < sensorNames.length)
      if ( rowIndex == columnIndex)
        dvalue = 1;
      else
        dvalue = 0;

    parameters[ rowIndex ][ columnIndex ] = dvalue;
    fireTableCellUpdated( rowIndex, columnIndex );
  }

  public void  setParameters (
      double[][]  parameters )
  {
    this.parameters = parameters;
    this.fireTableStructureChanged();
  }

  public double  getParameter (
      int rowIndex,
      int columnIndex )
  {
    return  parameters[rowIndex][columnIndex];
  }

  public void  setParameter (
      double  value,
      int rowIndex,
      int columnIndex )
  {
    if (Double.isNaN( value ))
      setValueAt( null, rowIndex, columnIndex );
    else
      setValueAt( new Double( value ), rowIndex, columnIndex );
  }
};

private ModelParameters  modelParameters = new ModelParameters();
private String[]  functionNames =
{ "biexponential", "hyperbolic", "log linear splice", "log (x + c)", "x + a log x
+ b" };
private LogicleFunction[]  logicleFunctions =
{ biexponential, hyperbolic, logLinearSplice, logPlusConstant, logPlus };
private DefaultComboBoxModel functionModel = new DefaultComboBoxModel(
functionNames );
private static final double  ln_10 = Math.log( 10 );
private final String  SPECTRUM_ERROR = "The data are not compatible with the
spectrum.";
private final int  FCS_TIMER_TICK = 200;
private final Random  random = new Random();

private void  interpolate (
    double[]  func,
    int  i,
    int  n,
    double  x )
{
  // exp((x+y)/2) = sqrt(exp(x)*exp(y))

  int  m = n / 2;
  double  y = Math.sqrt( func[i] * x );
```

```
  func[ i + m ] = y;

  if (m > 1 )
  {
    interpolate( func, i, m, y );
    interpolate( func, i + m, m, x );
  }

}

private void  shuffle (
    double[]  sequence )
{
  for (int  i = 0;  i < sequence.length;  ++i)
  {
    int j = i + random.nextInt( sequence.length - i );
    double  t = sequence[i];
    sequence[i] = sequence[j];
    sequence[j] = t;
  }
}

public void  invert (
    double[][]  matrix )
{
  int  row = 0, col = 0, n = matrix.length;
  int  pivot[] = new int[n];
  int  row_index[] = new int[n];
  int  col_index[] = new int[n];

  for (int  i = 0;  i < n;  ++i)
  {
    double  big = 0;
    for (int  j = 0;  j < n;  ++j)
    {
      if (pivot[j] != 1)
        for (int  k = 0;  k < n;  ++k)
        {
          if (pivot[k] == 0)
          {
            double  abs = Math.abs(matrix[j][k]);
            if (abs >= big)
            {
              big = abs;
              row = j;
              col = k;
            }
          }
          else if (pivot[k] > 1)
            throw  new IllegalArgumentException( "matrix is singular" );
        }
      }
      ++pivot[col];
      row_index[i] = row;
      col_index[i] = col;

      if (row != col)
        for (int  k = 0;  k < n;  ++k)
        {
          double  t = matrix[row][k];
          matrix[row][k] = matrix[col][k];
          matrix[col][k] = t;
```

```
    }

  if (matrix[col][col] == 0)
    throw  new IllegalArgumentException( "matrix is singular" );
  double inverse = 1/matrix[col][col];
  matrix[col][col] = 1;
  for (int  j = 0;  j < n;  ++j)
    matrix[col][j] *= inverse;
  for (int  j = 0;  j < n;  ++j)
    if (j != col)
    {
      double  t = matrix[j][col];
      matrix[j][col] = 0;
      for (int  k = 0;  k < n;  ++k)
        matrix[j][k] -= matrix[col][k] * t;
    }
 }

 for (int  i = n-1;  i >= 0;  --i)
   if (row_index[i] != col_index[i])
     for (int  j = 0;  j < n;  ++j)
     {
       double  t = matrix[j][row_index[i]];
       matrix[j][row_index[i]] = matrix[j][col_index[i]];
       matrix[j][col_index[i]] = t;
     }
}

double  root (
    double  b,
    double  w )
{
  double  Xlo = 0;
  double  Xhi = b;

  double  Flo = Double.NEGATIVE_INFINITY;
  double  Fhi = w * 2 * b;

  double d = (Xlo + Xhi)/2;
  double Dx = Math.abs( Xlo - Xhi );
  double Dx_last = Dx;

  double Fb = -2 * Math.log(b) + w*b;
  double F = 2 * Math.log(d) + w*d + Fb;
  double DF = 2/d + w;

  if (w == 0)
    return  b;

  for (int  i = 0;  i < 100;  ++i)
  {
    if (((d-Xhi)*DF-F)*((d-Xlo)*DF-F) >= 0
        || Math.abs( 2 * F ) > Math.abs( Dx_last * DF ))
    {
//        System.out.println( "bisection " + Dx );
      Dx = (Xhi - Xlo)/2;
      d = Xlo + Dx;
      if (d == Xlo)
        return  d;
    }
    else
    {
      Dx = F/DF;
```

```
//        System.out.println( "Newton " + Dx );
          double  t = d;
          d -= Dx;
          if (d == t)
            return  d;
        }
        if (Math.abs( Dx ) < 1E-12)
          return  d;
        Dx_last = Dx;

        F = 2 * Math.log(d) + w*d + Fb;
        DF = 2/d + w;
        if (F < 0)
        {
          Xlo = d;
          Flo = F;
        }
        else
        {
          Xhi = d;
          Fhi = F;
        }
      }

      throw  new IllegalStateException( "exceeded maximum iterations" );
    }

    void make_busy ()
    {
      spectrumButton.setEnabled( false );
      spectrumMenuItem.setEnabled( false );

      openButton.setEnabled( false );
      openMenuItem.setEnabled( false );

      saveButton.setEnabled( false );
      saveButton.setEnabled( false );

      progressBar.setEnabled( false );
    }

    void make_ready ()
    {
      spectrumButton.setEnabled( true );
      spectrumMenuItem.setEnabled( true );

      if (sensorNames != null)
      {
        openButton.setEnabled( true );
        openMenuItem.setEnabled( true );
      }

      if (fcs != null)
      {
        saveButton.setEnabled( true );
        saveMenuItem.setEnabled( true );
        menuScale.setEnabled( true );
        statusBar.setText( fcs.getFile().getName() );
      }
      else
      {
        saveButton.setEnabled( false );
        saveMenuItem.setEnabled( false );
```

```
    menuScale.setEnabled( false );
    statusBar.setText( "" );
  }

  progressBar.setValue( 0 );
  progressBar.setEnabled( false );
}

void spectrum_action (
    ActionEvent e )
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();

  fileChooser.setDialogTitle( "Load Spectrum File" );
  int returnVal = fileChooser.showOpenDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    make_busy();
    try
    {
      BufferedReader  br = new BufferedReader( new FileReader(
fileChooser.getSelectedFile() ) );
      br.readLine();
      br.readLine();
      StringTokenizer  st = new StringTokenizer( br.readLine(), "\t" );
      ArrayList  al = new ArrayList();
      while (st.hasMoreTokens())
        al.add( st.nextToken() );
      sensorNames = (String[]) al.toArray( stringArray );
      spectrumName = fileChooser.getSelectedFile().getName();

      double[][]  parameters = new double[sensorNames.length +
2][sensorNames.length];
      int  r;
      for (r = 0;  r < sensorNames.length;  ++r)
      {
        st = new StringTokenizer( br.readLine(), "\t" );
        for (int  c = 0;  c < sensorNames.length;  ++c)
          parameters[r][c] =  Double.parseDouble( st.nextToken() );
      }
      br.close();

      for (;  r < parameters.length;  ++r)
        for (int  c = 0;  c < sensorNames.length;  ++c)
          parameters[r][c] =  Double.NaN;
      modelParameters.setParameters( parameters );

      if (fcs != null)
        for (r = 0;  r < sensorNames.length;  ++r)
          if (fcs.getParameter( sensorNames[r] ) == null)
          {
            fcs = null;
            break;
          }
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
      make_ready();
    }
  }
```

```
boolean  get_compensation()
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();

  int  m = sensorNames.length;
  compensationMatrix = new double[m][m];
  for (int  i = 0;  i < m; ++i)
    for (int  j = 0;  j < m;  ++j)
      compensationMatrix[j][i] = modelParameters.getParameter( i, j );
  try
  {
    invert( compensationMatrix );
  }
  catch (IllegalArgumentException ex)
  {
    JOptionPane.showMessageDialog( this, "Spectrum Matrix is Singular", "Spectrum
Error",
        JOptionPane.ERROR_MESSAGE );
    return  false;
  }

  return  true;
}

FCSFile  fcs_data;
FCSHandler  fcs_handler;
IOException  fcs_exception;

Timer  fcs_timer = new Timer( FCS_TIMER_TICK, new ActionListener ()
{
  public void actionPerformed (
      ActionEvent  action )
  {
    if (fcs_handler != null)
      progressBar.setValue( fcs_handler.getEvent() );
  }
} );

void data_action(ActionEvent e)
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();

  fileChooser.setDialogTitle( "Load FCS Data File" );
  int returnVal = fileChooser.showOpenDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      File  f = fileChooser.getSelectedFile();

      fcs_data = new FCSFile( f );
      for (int  i = 0;  i < sensorNames.length;  ++i)
        if (fcs_data.getParameter( sensorNames[i] ) == null)
        {
          JOptionPane.showMessageDialog( this, SPECTRUM_ERROR,
              "Error", JOptionPane.ERROR_MESSAGE );
          return;
        }

      make_busy();
```

```
        progressBar.setMinimum( 0 );
        progressBar.setValue( 0 );
        progressBar.setMaximum( fcs_data.getTotal() );
        progressBar.setEnabled( true );
        statusBar.setText( fcs_data.getFile().getName() );

        new Thread( data_thread, "FCS reader" ).start();
        fcs_timer.start();
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
        System.exit( 1 );
      }
    }
  }

  Runnable  data_thread = new Runnable()
  {
    public void run ()
    {
      Thread.currentThread().setPriority( Thread.MIN_PRIORITY );
      fcs_exception = null;
      try
      {
        int  n = fcs_data.getTotal();
        int  m = sensorNames.length;
        int[][]  int_value = new int[fcs_data.getParameters()][n];
        double[][]  double_value = new double[m][n];

        fcs_handler = fcs_data.getInputIterator();
        for (int  i = 0;  fcs_handler.hasMoreEvents();  ++i)
          for (int  j = 0;  fcs_handler.hasMoreValues();  ++j)
            int_value[j][i] = fcs_handler.readValue();
        fcs_handler.close();

        for (int  i = 0;  i < m;  ++i)
        {
          FCSParameter  p = fcs_data.getParameter( sensorNames[i] );
          int[]  input_data = int_value[p.getIndex()-1];
          double[]  output_data = double_value[i];
          if (p.isLog())
          {
            double[]  antilog = new double[ p.getRange() ];
            antilog[0] = p.getMinimum();
            interpolate( antilog, 0, p.getRange(), p.getMaximum() );

            double[]  fuzz = new double[ 1<<FUZZY_BITS ];
            fuzz[0] = 1;
            interpolate( fuzz, 0, 1<<FUZZY_BITS, antilog[1]/antilog[0] );
            shuffle( fuzz );

            for (int  j = 0;  j < n;  ++j)
              output_data[j] = antilog[ input_data[j] ] * fuzz[ j &
((1<<FUZZY_BITS)-1) ];
          }
          else
          {
            double  g = p.getGain();
            for (int  j = 0;  j < n;  ++j)
              output_data[j] = input_data[j] / g;
          }
        }
```

```
      originalData = int_value;
      rawData = double_value;
    }
    catch (IOException ex)
    {
      fcs_exception = ex;
    }
    SwingUtilities.invokeLater( data_complete );
  }
};

Runnable  data_complete = new Runnable ()
{
  public void run ()
  {
    fcs_timer.stop();
    if (fcs_exception != null)
    {
      errorLabel1.setText( fcs_exception.getMessage() );
      errorLabel2.setText( fcs_data.getFile().getPath() );
      JOptionPane.showMessageDialog( ConverterFrame.this, errorMessage,
                                     "FCS Error", JOptionPane.ERROR_MESSAGE );

    }
    else
    {
      fcs = fcs_data;
      fcsTotal = fcs.getTotal();
    }
    make_ready();
  }
};

void save_action(ActionEvent e)
{
  if (!get_compensation())
    return;

  fileChooser.setDialogTitle( "Save New FCS Data File" );
  fileChooser.setSelectedFile( null );
  int returnVal = fileChooser.showSaveDialog( this );
  if(returnVal == JFileChooser.APPROVE_OPTION)
  {
    try
    {
      File  f = fileChooser.getSelectedFile();
      logicle = logicleFunctions[ functionChoice.getSelectedIndex() ];

      if (f.exists())
      {
        Object[]  message = { f, "already exists. Do you want to replace it?" };
        if (JOptionPane.OK_OPTION !=
            JOptionPane.showConfirmDialog( this, message, "Overwrite",
            JOptionPane.OK_CANCEL_OPTION, JOptionPane.WARNING_MESSAGE ))
          return;
      }

      make_busy();

      int  m = sensorNames.length;
      progress = 0;
      progressBar.setMinimum( 0 );
      progressBar.setMaximum( m + m * m  );
```

```
      progressBar.setValue( 0 );
      progressBar.setEnabled( true );
      statusBar.setText( "Working" );

      fcs_data = new FCSFile( f );
      fcs_data.setTextSegment( fcs.getTextSegment() );

      new Thread( xfrm_thread, "Transform" ).start();
    }
    catch (Exception ex)
    {
      ex.printStackTrace();
    }
  }
}

void  compensate_data()
{
  int  m = sensorNames.length;
  compensatedData = new double[m][fcsTotal];
  for (int  i = 0;  i < m;  ++i)
  {
    double[]  output_data = compensatedData[i];
    for (int  j = 0;  j < m;  ++j)
    {
      double[]  input_data = rawData[j];
      double  coefficient = compensationMatrix[i][j];
      for (int  k = 0;  k < fcsTotal;  ++k)
        output_data[k] += input_data[k] * coefficient;
      ++progress;
      SwingUtilities.invokeLater( progress_made );
    }
  }
}

Runnable  xfrm_thread = new Runnable()
{
  public void run ()
  {
    try
    {
      compensate_data();

      int  m = sensorNames.length;
      newData = new int[m][fcsTotal];
      for (int  i = 0;  i < m;  ++i)
      {
        FCSParameter  oldp = fcs.getParameter( sensorNames[i] );
        FCSParameter  newp = fcs_data.addParameter();
        newp.setAttribute( "$P", "N", "["+sensorNames[i]+"]" );
        newp.setAttribute( "$P", "S", oldp.getAttribute( "$P", "S" ) );
        newp.setBits( oldp.getBits() );
        newp.setRange( oldp.getRange() );
        newp.setDecades( oldp.getDecades(), oldp.getMinimum() );
        newp.setGain( oldp.getGain() );

        double[]  input_data = compensatedData[i];
        int[]  transformed_data = newData[i];
        if (newp.isLog())
        {
          double[]  lookup = logicle.transform( i, newp );
          newp.setDecades( logicle.decades, logicle.minimum );
          for (int  j = 0;  j < fcsTotal;  ++j)
```

```
        {
          int  k = Arrays.binarySearch( lookup, input_data[j] );
          if (k < 0)
            transformed_data[j] = -k-1;
          else
            transformed_data[j] = k;
        }
      }
      else
      {
        double  g = newp.getGain();
        for (int  j = 0;  j < fcsTotal;  ++i)
          transformed_data[j] = (int) Math.round( input_data[j] * g );
      }
      ++progress;
      SwingUtilities.invokeLater( progress_made );
    }

    compensatedData = null;
    SwingUtilities.invokeLater( xfrm_complete );
  }
  catch (Exception ex)
  {
    ex.printStackTrace();
  }
 }
};

Runnable  progress_made = new Runnable ()
{
 public void run ()
 {
   progressBar.setValue( progress );
 }
};

Runnable  xfrm_complete = new Runnable ()
{
 public void  run ()
 {
   progressBar.setMinimum( 0 );
   progressBar.setValue( 0 );
   progressBar.setMaximum( fcs_data.getTotal() );
   progressBar.setEnabled( true );
   statusBar.setText( fcs_data.getFile().getName() );

   new Thread( save_thread, "FCS writer" ).start();
   fcs_timer.start();
 }
};

Runnable  save_thread = new Runnable()
{
 public void run ()
 {
   Thread.currentThread().setPriority( Thread.MIN_PRIORITY );
   fcs_exception = null;
   try
   {
     fcs_handler = fcs_data.getOutputIterator();
     for (int  i = 0;  fcs_handler.hasMoreEvents();  ++i)
     {
       for (int  j = 0;  j < originalData.length;  ++j)
```

```
            fcs_handler.writeValue( originalData[j][i] );
          for (int  j = 0;  j < newData.length;  ++j)
            fcs_handler.writeValue( newData[j][i] );
        }
        fcs_handler.close();
      }
      catch (IOException ex)
      {
        ex.printStackTrace();
        fcs_exception = ex;
      }
      SwingUtilities.invokeLater( save_complete );
    }
  };

  Runnable  save_complete = new Runnable ()
  {
    public void run ()
    {
      fcs_timer.stop();
      if (fcs_exception != null)
      {
        errorLabel1.setText( fcs_exception.getMessage() );
        errorLabel2.setText( fcs_data.getFile().getPath() );
        JOptionPane.showMessageDialog( ConverterFrame.this, errorMessage,
                                       "FCS Error", JOptionPane.ERROR_MESSAGE );

      }
      make_ready();
    }
  };

  void function_action(ActionEvent e)
  {
    if (spectrumTable.isEditing())
      spectrumTable.getCellEditor().stopCellEditing();

    for (int  i = 0;  i < sensorNames.length;  ++i)
      for (int  j = 0;  j < this.N_PARAMETERS;  ++j)
        modelParameters.setValueAt( null, sensorNames.length + j, i );

    logicle = logicleFunctions[ functionChoice.getSelectedIndex() ];
  }

  void nomogram_action()
  {
    int columnIndex = spectrumTable.getSelectedColumn();
    if (columnIndex < 0 || fcs == null)
      return;
    FCSParameter  p;
    try
    {
      p = fcs.getParameter( sensorNames[columnIndex] );
    }
    catch (IOException ex)
    {
      return;
    }

    if (p.isLog())
    {
      LogicleFunction  function = logicleFunctions[
functionChoice.getSelectedIndex() ];
      double[]  right = function.transform( columnIndex, p );
```

```
    double[]  left = new double[ p.getRange() ];
    left[0] = function.minimum;
    interpolate( left, 0, left.length, function.maximum );
    axisDialog.nomoRelative = new double[axisDialog.nomoLabel.length][2];
    for (int  i = 0;  i < axisDialog.nomoLabel.length;  ++i)
    {
      double  x = Double.parseDouble( axisDialog.nomoLabel[i] );
      int  k = Arrays.binarySearch( left, x );
      if (k < 0)
        axisDialog.nomoRelative[i][0] = (double) (-k-1) / (double) left.length;
      else
        axisDialog.nomoRelative[i][0] = (double) k / (double) left.length;

      k = Arrays.binarySearch( right, x );
      if (k < 0)
        axisDialog.nomoRelative[i][1] = (double) (-k-1) / (double) right.length;
      else
        axisDialog.nomoRelative[i][1] = (double) k / (double) right.length;
    }
    axisDialog.repaint();
  }
  else
  {
    double  g = p.getGain();
  }
}

void help_action(ActionEvent e)
{
  helpDialog.setLocationRelativeTo( this );
  helpDialog.setVisible( true );
}

void help_dismiss(ActionEvent e)
{
  helpDialog.setVisible( false );
}

void about_action(ActionEvent e)
{
  JOptionPane.showMessageDialog( this, aboutMessage, "About Logicle",
JOptionPane.PLAIN_MESSAGE );
}
public void tableChanged(TableModelEvent e)
{
  /**@todo Implement this javax.swing.event.TableModelListener method*/
  throw new java.lang.UnsupportedOperationException("Method tableChanged() not yet
implemented.");
}

void view_action(ActionEvent e)
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();
  axisDialog.setVisible( true );
  axisDialog.toFront();
}

void scale_action(ActionEvent e)
{
  if (spectrumTable.isEditing())
    spectrumTable.getCellEditor().stopCellEditing();
  if (fcs == null)
```

```
      return;
    fileChooser.setDialogTitle( "Save Logicle Scale" );
    String  fn = fcs.getFile().getName();
    if (fn.toUpperCase().endsWith( ".FCS" ))
      fn = fn.substring( 0, fn.length() - ".FCS".length() );
    fileChooser.setSelectedFile( new File( fcs.getFile().getParentFile(), fn +
".logicle" ) );
    int returnVal = fileChooser.showSaveDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        File  f = fileChooser.getSelectedFile();
        logicle = logicleFunctions[ functionChoice.getSelectedIndex() ];

        if (f.exists())
        {
          Object[]  message = { f, "already exists. Do you want to replace it?" };
          if (JOptionPane.OK_OPTION !=
              JOptionPane.showConfirmDialog( this, message, "Overwrite",
              JOptionPane.OK_CANCEL_OPTION, JOptionPane.WARNING_MESSAGE ))
            return;
        }

        int c = spectrumTable.getSelectedColumn();
        if (c < 0)
          return;
        FCSParameter  p = fcs.getParameter( sensorNames[c] );

        double[] scale = logicle.transform( c, p );
        PrintWriter  pw = new PrintWriter( new FileOutputStream( f ) );
        for (int  i = 0;  i < scale.length;  ++i)
          pw.println( scale[i] );
        pw.close();
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  }

  void new_scale_action(ActionEvent e)
  {
    if (spectrumTable.isEditing())
      spectrumTable.getCellEditor().stopCellEditing();
    if (fcs == null)
      return;

    fileChooser.setDialogTitle( "Save Logicle Scales" );
    String  fn = fcs.getFile().getName();
    if (fn.toUpperCase().endsWith( ".FCS" ))
      fn = fn.substring( 0, fn.length() - ".FCS".length() );
    fileChooser.setSelectedFile( new File( fcs.getFile().getParentFile(), fn +
".logicle" ) );
    int returnVal = fileChooser.showSaveDialog( this );
    if(returnVal == JFileChooser.APPROVE_OPTION)
    {
      try
      {
        File  f = fileChooser.getSelectedFile();
        logicle = logicleFunctions[ functionChoice.getSelectedIndex() ];
```

```
    if (f.exists())
    {
      Object[]  message = { f, "already exists. Do you want to replace it?" };
      if (JOptionPane.OK_OPTION !=
          JOptionPane.showConfirmDialog( this, message, "Overwrite",
          JOptionPane.OK_CANCEL_OPTION, JOptionPane.WARNING_MESSAGE ))
        return;
    }

    PrintWriter  pw = new PrintWriter( new FileOutputStream( f ) );
    for (int  i = 0;  i < sensorNames.length;  ++i)
    {
      FCSParameter  p = fcs.getParameter( sensorNames[i] );

      pw.println( spectrumName+" ["+sensorNames[i]+"] <"+sensorNames[i]+">" );
      pw.println( p.getRange() + 1 );
      double[] scale = logicle.transform( i, p );
      for (int  j = 0;  j < scale.length;  ++j)
        pw.println( scale[j] );
      pw.println( p.getMaximum() );
    }
    pw.close();
   }
   catch (Exception ex)
   {
     ex.printStackTrace();
   }
  }
}

private double autoscaleParameter;

void  autoscale_action ()
{
  if (!get_compensation())
    return;

  logicle = logicleFunctions[ functionChoice.getSelectedIndex() ];

  try
  {
    make_busy();

    int  m = sensorNames.length;
    progress = 0;
    progressBar.setMinimum( 0 );
    progressBar.setMaximum( m + m * m  );
    progressBar.setValue( 0 );
    progressBar.setEnabled( true );
    statusBar.setText( "working" );

    new Thread( autoscale_thread, "Autoscale" ).start();
  }
  catch (Exception ex)
  {
    ex.printStackTrace();
  }
 }

 Runnable  autoscale_thread = new Runnable()
 {
   public void run ()
```

```
    {
      try
      {
        compensate_data();

        int  m = sensorNames.length;
        for (int  i = 0;  i < m;  ++i)
        {
          double[]  sample_data = compensatedData[i];
          Arrays.sort( sample_data );
          int  scale_total;
          for (scale_total = 0; sample_data[scale_total] < 0 && scale_total <
fcsTotal; ++scale_total);
          double  value = 0;
          if (scale_total > 0)
            value = -sample_data[(int) Math.round( autoscaleParameter * scale_total
)];

          logicle.automaticScale( i, fcs.getParameter( sensorNames[i] ), value );

          ++progress;
          SwingUtilities.invokeLater( progress_made );
        }

        compensatedData = null;
        SwingUtilities.invokeLater( autoscale_complete );
      }
      catch (Exception ex)
      {
        ex.printStackTrace();
      }
    }
  };

  Runnable  autoscale_complete = new Runnable ()
  {
    public void  run ()
    {
      nomogram_action();
      make_ready();
    }
  };
  private JMenuItem jMenuItem6 = new JMenuItem();

  void automatic_action_1_neg(ActionEvent e)
  {
    autoscaleParameter = .01;
    autoscale_action();
  }

  void automatic_action_5_neg(ActionEvent e)
  {
    autoscaleParameter = .05;
    autoscale_action();
  }

  void automatic_action_0_neg(ActionEvent e)
  {
    autoscaleParameter = 0;
    autoscale_action();
  }
}
```

```
package edu.stanford.facs.loglike;

import javax.swing.*;
import java.awt.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class DataConverter {
  private boolean packFrame = false;

  //Construct the application
  public DataConverter() {
    ConverterFrame frame = new ConverterFrame();
    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
    frame.setVisible(true);
  }
  //Main method
  public static void main(String[] args) {
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    new DataConverter();
  }
}
```

```
package edu.stanford.facs.loglike;

import javax.swing.*;
import java.awt.*;

/**
 * <p>Title: Log Like Data Transform</p>
 * <p>Description: </p>
 * <p>Copyright: Copyright (c) 2002</p>
 * <p>Company: Stanford University</p>
 * @author Wayne A. Moore
 * @version 1.0
 */

public class DataConverter {
  private boolean packFrame = false;

  //Construct the application
  public DataConverter() {
    ConverterFrame frame = new ConverterFrame();
    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame) {
      frame.pack();
    }
    else {
      frame.validate();
    }
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2, (screenSize.height -
frameSize.height) / 2);
    frame.setVisible(true);
  }
  //Main method
  public static void main(String[] args) {
    try {
      UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    new DataConverter();
  }
}
```

```
<html>
<head>
</head>
<body>
<center><h1>Logicle Fluorescence
Compensation</h1></center>
<p>
This program computes compensated
fluorescence data from high dynamic range raw data
that are typically
plotted on a logarithmic scale.
The <b>linear</b> process of spectral
overlap correction of raw fluorescence values or "fluorescence
compensation"
naturally produces small and negative values that are
not handled well by <b>log</b> scaling.
This program extends
<code><b>FCS</b></code> data files to include various
<b>log</b>
<i>like</i> representations of the data, which are intended to
have
better properties for small and negative values.
</p>
<p>
The
transformations require as input a <b>spectrum</b> matrix exported
from <b>FlowJo<font size="-2"><sup>TM</sup></font></b>
and one or two
additional parameters.
For example for a simple two color spectrum
<code><b>Fluor</b></code> and
<code><b>PhyEry</b></code>
<center>
<table border="1" cellpadding="4"
cellspacing="2">
<col width="2*" rules="cols">
<col width="1*"
span="2">
<tr><th></th><th><code><b>Fluor</b></code></th><th><code><b>
PhyEry</b></code></th></tr>
<tr><th align="right" style="color:
blue"><code><b>Fluor</b></code>
Control</th><td>1.0</td><td>0.1983</td></tr>
<tr><th align="right"
style="color: blue"><code><b>PhyEry</b></code>
Control</th><td>0.0088568</td><td>1.0</td></tr>
<tr><th align="right"
style="color: red">Parameter 1</th><td></td><td></td></tr>
<tr><th
align="right" style="color: red">Parameter
2</th><td></td><td></td></tr>
</table>
</center>
The first
<code><b>n</b></code> rows are the ordinary (normalized) spillover
coefficients for each of
the dyes used.
The meaning of the two
parameters varies depending on the function choosen.
You can load a
new spectrum at any time.
</p>
<p>
```

Once you've loaded a spectum you
can open an <code>FCS</code> data file
for which the spectrum
is appropriate, i.e., it contains data for all the fluorochromes
listed.
Finally you can save an extended <code>FCS</code> file
that contains the new compensated data
as well as any previous
contents.
For example, if you initially had a file with raw data for
<code>Fluor</code> and <code>PhyEry</code>,
then new
parameters <code>[Fluor]</code> and
<code>[PhyEry]</code> with the transformed data will be
added.
Note that the compensated values are computed independantly of
FlowJo^{TM}
and that
FlowJo^{TM} compensated
parameters <code><Fluor></code>
and
<code><PhyEry></code> will be preserved unchanged if
present.

<blockquote>
<font size="+1" style="indent:
10px">
You can alter the function parameters to adjust them to your
data and you can alter
the spectral overlap values if you wish to
tinker with the
compensation.

</blockquote>

<h2>Functions
Available</h2>

The first two functions are parameterized by a
single value that characterizes
the desired range (in the units
encoded in the <code>FCS</code> file) of the
bipolar
linear region of the scale.
For both functions a FlowJo^{TM} plot of the transformed data
the
negative linear boundry is the minimum value plotted, the zeroes will
fall
at one factor of <code>e</code> above the minimum and the
linear boundry
is at <code>e²</code>.

<h3>log
linear splice</h3>

The data are transformed linearly up to the
parameter value and logarithmicly beyond it.
The splice is made so
that it is continuous in the first derivative although not in the
second.

```
The scale values at the boundry and above are identical to
the normal values.
</p>
<h3>hyperbolic</h3>
<p>
The data are scaled
to bring the linear region to <code>|x| &lt; 1</code>
then
<code><b>sinh</b></code> is taken and scaled in turn to produce
the ordinary
logarithm for large values.
The <code><b>sinh</b></code>
function is odd and linear for small values and
exponential at large
values (more than a decade above the linear threshold),
while the
derivatives are as well behaved as they get.
</p>
<h3>x + a log x +
b</h3>
<p>
A normal antilog table is computed for the data then the
values in the table
are offset by a linear function.
With suitable
parameters, the small values will be approximately linear and
large
values will be offset by insignificant amounts.
The first parameter
gives the approximate linear value that will be plotted at
one decade
above the minimum.
The second (optional) parameter can be used to
displace the linearized region
to include negative
numbers.
</p>
<hr>
<center><i><font size="-1">
Copyright &copy; 2002
by The Board of Trustees<br>
of the Leland Stanford Jr.
University
</font></i></center>
</body>
</html>
```

```
<html>
<head>
</head>
<body>
<center><h1>Logicle Fluorescence Compensation</h1></center>
<p>
This program computes compensated fluorescence data from high dynamic range raw data
that are typically plotted on a logarithmic scale.
The <b>linear</b> process of spectral overlap correction of raw fluorescence values
or "fluorescence compensation"
naturally produces small and negative values that are not handled well by <b>log</b>
scaling.
This program extends <code><b>FCS</b></code> data files to include various
<b>log</b> <i>like</i> representations of the data, which are intended to
have better properties for small and negative values.
</p>
<p>
The transformations require as input a <b>spectrum</b> matrix exported from
<b>FlowJo<font size="-2"><sup>TM</sup></font></b>
and one or two additional parameters.
For example for a simple two color spectrum <code><b>Fluor</b></code> and
<code><b>PhyEry</b></code>
<center>
<table border="1" cellpadding="4" cellspacing="2">
<col width="2*" rules="cols">
<col width="1*" span="2">
<tr><th></th><th><code><b>Fluor</b></code></th><th><code><b>PhyEry</b></code></th></
tr>
<tr><th align="right" style="color: blue"><code><b>Fluor</b></code>
Control</th><td>1.0</td><td>0.1983</td></tr>
<tr><th align="right" style="color: blue"><code><b>PhyEry</b></code>
Control</th><td>0.0088568</td><td>1.0</td></tr>
<tr><th align="right" style="color: red">Parameter 1</th><td></td><td></td></tr>
<tr><th align="right" style="color: red">Parameter 2</th><td></td><td></td></tr>
</table>
</center>
The first <code><b>n</b></code> rows are the ordinary (normalized) spillover
coefficients for each of
the dyes used.
The meaning of the two parameters varies depending on the function choosen.
You can load a new spectrum at any time.
</p>
<p>
Once you've loaded a spectum you can open an <code><b>FCS</b></code> data file
for which the spectrum is appropriate, i.e., it contains data for all the
fluorochromes listed.
Finally you can save an extended <code><b>FCS</b></code> file that contains the new
compensated data
as well as any previous contents.
For example, if you initially had a file with raw data for <code><b>Fluor</b></code>
and <code><b>PhyEry</b></code>,
then new parameters <code><b>[Fluor]</b></code> and <code><b>[PhyEry]</b></code>
with the transformed data will be added.
Note that the compensated values are computed independantly of <b>FlowJo<font
size="-2"><sup>TM</sup></font></b>
and that <b>FlowJo<font size="-2"><sup>TM</sup></font></b> compensated parameters
<code><b>&lt;Fluor&gt;</b></code>
and <code><b>&lt;PhyEry&gt;</b></code> will be preserved unchanged if present.
</p>
<p>
<blockquote>
<font size="+1" style="indent: 10px">
You can alter the function parameters to adjust them to your data and you can alter
```

```
the spectral overlap values if you wish to tinker with the compensation.
</font>
</blockquote>
</p>
<p>
If you first select one of the data columns and then choose <b>View</b> from the
<b>Scale</b> menu a side by side comparison of the default <b>FlowJo<font
size="-2"><sup>TM</sup></font></b> display scale and
the <code><b>Logicle</b></code> scale is displayed.
If you choose <b>Save</b> from the <b>Scale</b> menu the scale for that parameter
will be
saved in a file that <b>FlowJo<font size="-2"><sup>TM</sup></font></b> can use to
display the scale correctly.
Finally, if you choose <b>Automatic</b> then <code><b>Logicle</b></code> will
attempt to
set the scale appropriately for the data according to the chosen criterea.
</p>

<h2>Functions Available</h2>

<h3>Biexponential</h3>
<p>
The biexponential transformation uses a function of the form
<center> f(x) = a e<sup>bx</sup> - c e<sup>-dx</sup></center>
to transform the data.
The first parameter is the number of decades (onto the original scale) where
the transition from the negative to the positive exponential occurs.
By default the same value is used to set the range of negative values shown.
The second parameter is optional and if present, indicates the number of extra
decades worth of space added to negative range.
</p>

<h3>log (x+c) </h3>
<p>
Adds a constant to the value and then takes the ordinary logarithm
</p>
<p>
The next two functions are parameterized by a single value that characterizes
the desired range (in the units encoded in the <code><b>FCS</b></code> file) of the
<b>bipolar linear</b> region of the scale.
For both functions a <b>FlowJo<font size="-2"><sup>TM</sup></font></b> plot of the
transformed data
the negative linear boundry is the minimum value plotted, the zeroes will fall
at one factor of <b><code>e</code></b> above the minimum and the linear boundry
is at <b><code>e<sup>2</sup></code></b>.
</p>
<h3>log linear splice</h3>
<p>
The data are transformed linearly up to the parameter value and logarithmicly beyond
it.
The splice is made so that it is continuous in the first derivative although not in
the second.
The scale values at the boundry and above are identical to the normal values.
</p>
<h3>hyperbolic</h3>
<p>
The data are scaled to bring the linear region to <code>|x| &lt; 1</code> then
<code><b>sinh</b></code> is taken and scaled in turn to produce the ordinary
logarithm for large values.
The <code><b>sinh</b></code> function is odd and linear for small values and
exponential at large values (more than a decade above the linear threshold),
while the derivatives are as well behaved as they get.
</p>
```

```
<h3>x + a log x + b</h3>
<p>
A normal antilog table is computed for the data then the values in the table
are offset by a linear function.
With suitable parameters, the small values will be approximately linear and
large values will be offset by insignificant amounts.
The first parameter gives the approximate linear value that will be plotted at
one decade above the minimum.
The second (optional) parameter can be used to displace the linearized region
to include negative numbers.
</p>
<hr>
<center><i><font size="-1">
Copyright &copy; 2002 by The Board of Trustees<br>
of the Leland Stanford Jr. University
</font></i></center>
</body>
</html>
```